Microsoft Certified:
Azure Administrator Associate

AZ-104: Prerequisites for Azure administrators

Configure Azure resources with tools

Azure Administrators use tools to interact with the cloud environment and complete such tasks as:

- Deploying dozens or hundreds of resources at a time.
- Configuring individual services using scripts.
- Viewing rich reports across usage, health, costs, and more.

You must select and use a tooling option. Your choices can include the Azure portal, Azure PowerShell, Azure CLI, or Azure Cloud Shell.

Azure Cloud Shell is an interactive, browser-accessible shell for managing Azure resources. Linux users can opt for a Bash experience, while Windows users can opt for PowerShell. Features:

- Times out after 20 minutes without interactive activity.
- Requires a resource group, storage account, and Azure File share.
- Uses the same Azure file share for both Bash and PowerShell.
- Is assigned to one machine per user account.
- Persists $HOME using a 5-GB image held in your file share.
- Permissions are set as a regular Linux user in Bash.

Azure PowerShell is a module that you add to Windows PowerShell or PowerShell Core to enable you to connect to your Azure subscription and manage resources. Azure PowerShell requires PowerShell to function. PowerShell provides services such as the shell window and command parsing. Azure PowerShell adds the Azure-specific commands.

```
New-AzVm `
    -ResourceGroupName "CrmTestingResourceGroup" `
    -Name "CrmUnitTests" `
    -Image "UbuntuLTS"
    ...
```

Azure PowerShell is also available two ways: inside a browser via the Azure Cloud Shell, or with a local installation on Linux, macOS, or the Windows operating system. In both cases, you have two modes from which to choose: you can use it in interactive mode in which you manually issue one command at a time, or in scripting mode where you execute a script that consists of multiple commands.

What is the Az module? Az is the formal name for the Azure PowerShell module containing cmdlets to work with Azure features.

Azure CLI is a command-line program to connect to Azure and execute administrative commands on Azure resources. It runs on Linux, macOS, and Windows, and allows

administrators and developers to execute their commands through a terminal, command-line prompt, or script instead of a web browser. For example, to restart a VM, you would use a command such as the following:

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

You can also use Azure CLI from a browser through Azure Cloud Shell. In both cases, Azure CLI can be used interactively or through scripts.

Commands in the CLI are structured in groups and subgroups. Each group represents a service provided by Azure, and the subgroups divide commands for these services into logical groupings. For example, the storage group contains subgroups including account, blob, share, and queue.

So, how do you find the particular commands you need? One way is to use az find. For example, if you want to find commands that might help you manage a storage blob, you can use the find command:

```
az find blob
```

If you already know the name of the command you want, the --help argument for that command will get you more detailed information on the command, and for a command group, a list of the available subcommands.

Use Azure Resource Manager

Your company is beginning to create resources in Azure. There is no organizational plan for standardizing the effort. There have been several instances where critical resources were inadvertently deleted. It is difficult to determine who owns which resource.

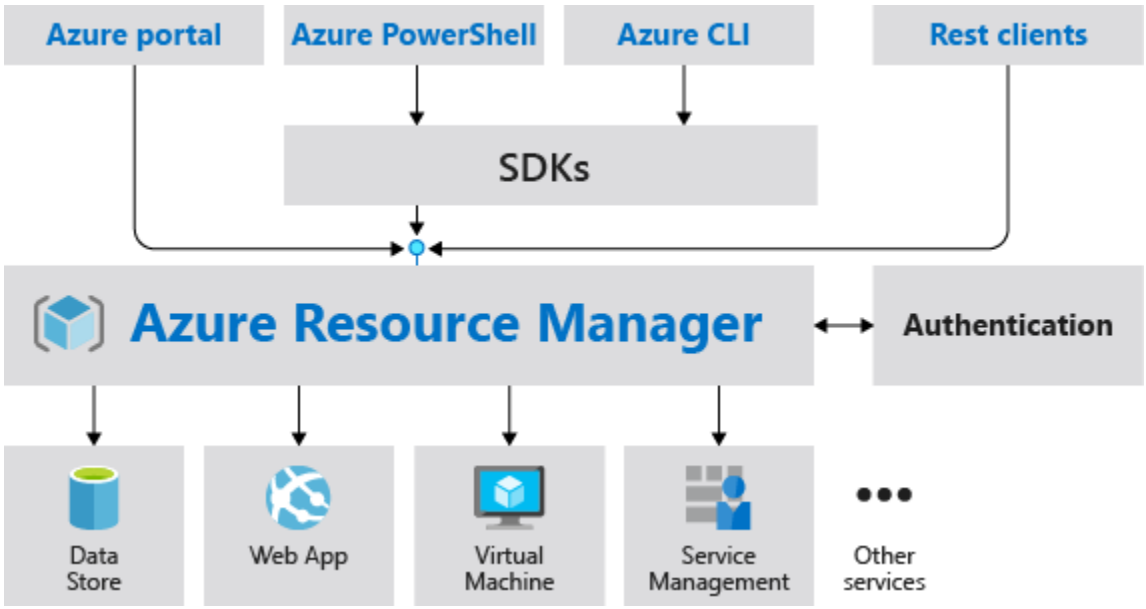
You need to use resource groups to organize the company's Azure resources.

The infrastructure for your application is typically made up of many components – maybe a virtual machine, storage account, and virtual network, or a web app, database, database server, and third-party services. These components are not separate entities, instead they are related and interdependent parts of a single entity. You want to deploy, manage, and monitor them as a group.

You use a template for deployment and that template can work for different environments such as testing, staging, and production. Azure Resource Manager provides security, auditing, and tagging features to help you manage your resources after deployment.

The following image shows how all the tools interact with the same Azure Resource Manager API. The API passes requests to the Azure Resource Manager service, which authenticates and

authorizes the requests. Azure Resource Manager then routes the requests to the appropriate resource providers.



Azure Resource Manager provides several benefits:

- You can deploy, manage, and monitor all the resources for your solution as a group, rather than handling these resources individually.
- You can repeatedly deploy your solution throughout the development lifecycle and have confidence your resources are deployed in a consistent state.
- You can manage your infrastructure through declarative templates rather than scripts.
- You can define the dependencies between resources so they're deployed in the correct order.
- You can apply access control to all services in your resource group because Role-Based Access Control (RBAC) is natively integrated into the management platform.
- You can apply tags to resources to logically organize all the resources in your subscription.
- You can clarify your organization's billing by viewing costs for a group of resources sharing the same tag.

The following suggestions help you take full advantage of Azure Resource Manager when working with your solutions.

- Define and deploy your infrastructure through the declarative syntax in Azure Resource Manager templates, rather than through imperative commands.
- Define all deployment and configuration steps in the template. You should have no manual steps for setting up your solution.
- Run imperative commands to manage your resources, such as to start or stop an app or machine.

- Arrange resources with the same lifecycle in a resource group. Use tags for all other organizing of resources.

resource provider - A service that supplies the resources you can deploy and manage through Resource Manager. Each resource provider offers operations for working with the resources that are deployed. Some common resource providers are Microsoft.Compute, which supplies the virtual machine resource, Microsoft.Storage, which supplies the storage account resource, and Microsoft.Web, which supplies resources related to web apps. The name of a resource type is in the format: {resource-provider}/{resource-type}. For example, the key vault type is Microsoft.KeyVault/vaults.

template - A JavaScript Object Notation (JSON) file that defines one or more resources to deploy to a resource group. It also defines the dependencies between the deployed resources. The template can be used to deploy the resources consistently and repeatedly.

declarative syntax - Syntax that lets you state "Here is what I intend to create" without having to write the sequence of programming commands to create it. The Resource Manager template is an example of declarative syntax.

There are a few rules for resource groups.

- Resources can only exist in one resource group.
- Resource Groups cannot be renamed.
- Resource Groups can have resources of many different types (services).
- Resource Groups can have resources from many different regions.
- All the resources in your group should share the same lifecycle. You deploy, update, and delete them together.
- You can move a resource from one resource group to another group. Limitations do apply to moving resources
- A resource can interact with resources in other resource groups; when the two resources are related but don't share the same lifecycle

Resource Manager locks allow organizations to put a structure in place that prevents the accidental deletion of resources in Azure.

- You can associate the lock with a subscription, resource group, or resource.
- Locks are inherited by child resources.

There are two types of resource locks.

- Read-Only locks, which prevent any changes to the resource.
- Delete locks, which prevent deletion.

Sometimes you may need to move resources to either a new subscription or a new resource group in the same subscription. When moving resources, both the source group and the target group are locked during the operation. Write and delete operations are blocked on the resource groups until the move completes. Locks don't mean the resources aren't available.

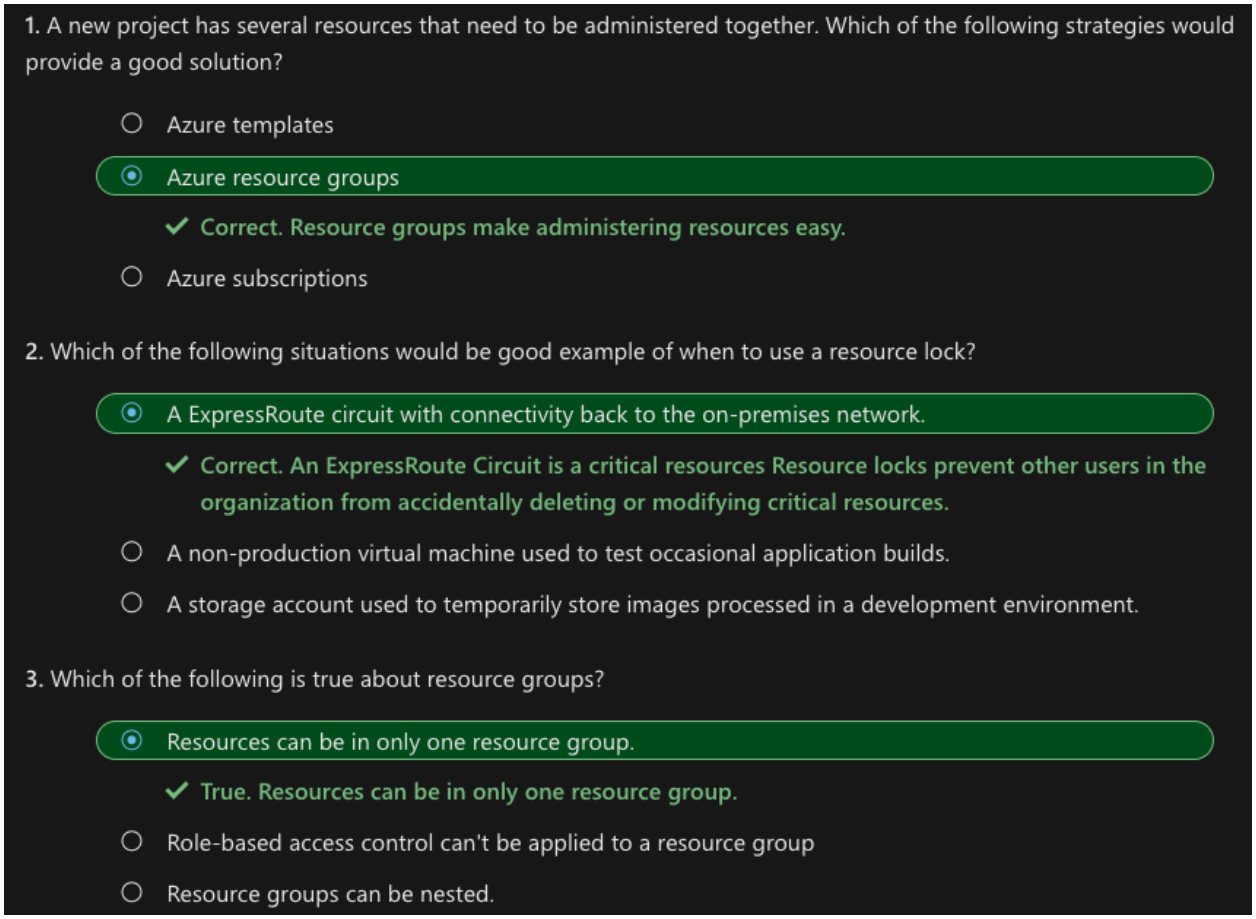
Use caution when deleting a resource group. Deleting a resource group deletes all the resources contained within it. That resource group might contain resources that resources in other resource groups depend on.

Using PowerShell to delete resource groups:

Remove-AzResourceGroup -Name "ContosoRG01"

Azure lets you view resource usage against limits. This is helpful to track current usage, and plan for future use.

- The limits shown are the limits for your subscription.
- When you need to increase a default limit, there is a Request Increase link.
- All resources have a maximum limit listed in Azure limits.
- If you are at the maximum limit, the limit can't be increased.



Configure resources with Azure Resource Manager templates

Your company needs to ensure virtual machine deployments are consistent across the organization. You use Azure Resource Manager templates to deploy resources including virtual machines.

An Azure Resource Manager template precisely defines all the Resource Manager resources in a deployment. You can deploy a Resource Manager template into a resource group as a single operation.

Template benefits

- Templates improve consistency. Resource Manager templates provide a common language for you and others to describe your deployments. Regardless of the tool or SDK that you use to deploy the template, the structure, format, and expressions inside the template remain the same.
- Templates help express complex deployments. Templates enable you to deploy multiple resources in the correct order. For example, you wouldn't want to deploy a virtual machine prior to creating an operating system (OS) disk or network interface. Resource Manager maps out each resource and its dependent resources, and creates dependent resources first. Dependency mapping helps ensure that the deployment is carried out in the correct order.
- Templates reduce manual, error-prone tasks. Manually creating and connecting resources can be time consuming, and it's easy to make mistakes. Resource Manager ensures that the deployment happens the same way every time.
- Templates are code. Templates express your requirements through code. Think of a template as a type of Infrastructure as Code that can be shared, tested, and versioned similar to any other piece of software. Also, because templates are code, you can create a "paper trail" that you can follow. The template code documents the deployment. Most users maintain their templates under some kind of revision control, such as GIT. When you change the template, its revision history also documents how the template (and your deployment) has evolved over time.
- Templates promote reuse. Your template can contain parameters that are filled in when the template runs. A parameter can define a username or password, a domain name, and so on. Template parameters enable you to create multiple versions of your infrastructure, such as staging and production, while still using the exact same template.
- Templates are linkable. You can link Resource Manager templates together to make the templates themselves modular. You can write small templates that each define a piece of a solution, and then combine them to create a complete system.
- Templates simplify orchestration. You only need to deploy the template to deploy all of your resources. Normally this would take multiple operations.

A Resource Manager template can contain sections that are expressed using JSON notation, but aren't related to the JSON language itself:

```
{
                                                    "$schema":
"http://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
   "contentVersion": "",
```

```
    "parameters": {},
    "variables": {},
    "functions": [],
    "resources": [],
    "outputs": {}
}
```

Element name	Required	Description
$schema	Yes	Location of the JSON schema file that describes the version of the template language. Use the URL shown in the preceding example.
contentVersion	Yes	Version of the template (such as 1.0.0.0). You can provide any value for this element. Use this value to document significant changes in your template. This value can be used to make sure that the right template is being used.
parameters	No	Values that are provided when deployment is executed to customize resource deployment.
variables	No	Values that are used as JSON fragments in the template to simplify template language expressions.
functions	No	User-defined functions that are available within the template.
resources	Yes	Resource types that are deployed or updated in a resource group.
outputs	No	Values that are returned after deployment.

In the parameters section of the template, you specify which values you can input when deploying the resources. The available properties for a parameter are:

```
"parameters": {
    "<parameter-name>" : {
        "type" : "<type-of-parameter-value>",
        "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>"
      }
    }
}
```

Azure Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse.

You can use Bicep instead of JSON to develop your Azure Resource Manager templates (ARM templates). The JSON syntax to create an ARM template can be verbose and require complicated expressions. Bicep syntax reduces that complexity and improves the development experience. Bicep is a transparent abstraction over ARM template JSON and doesn't lose any of the JSON template capabilities.

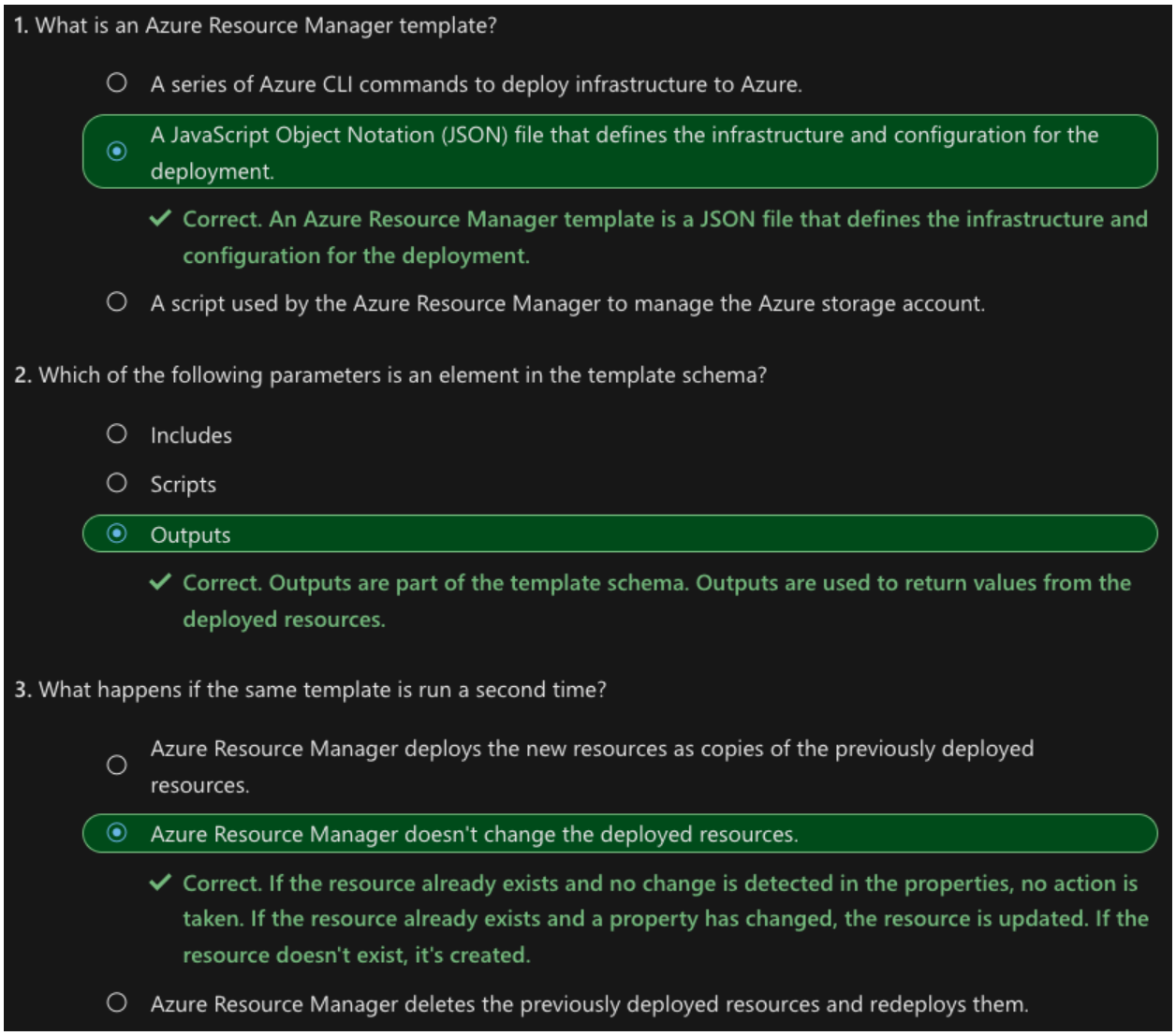
How does Bicep work? When you deploy a resource or series of resources to Azure, the tooling that's built into Bicep converts your Bicep template into a JSON template. This process is known as transpilation. Transpilation is the process of converting source code written in one language into another language.

Bicep provides many improvements over JSON for template authoring, including:

- Simpler syntax: Bicep provides a simpler syntax for writing templates. You can reference parameters and variables directly, without using complicated functions. String interpolation is used in place of concatenation to combine values for names and other items. You can reference the properties of a resource directly by using its symbolic name instead of complex reference statements. These syntax improvements help both with authoring and reading Bicep templates.
- Modules: You can break down complex template deployments into smaller module files and reference them in a main template. These modules provide easier management and greater reusability.
- Automatic dependency management: In most situations, Bicep automatically detects dependencies between your resources. This process removes some of the work involved in template authoring.
- Type validation and IntelliSense: The Bicep extension for Visual Studio Code features rich validation and IntelliSense for all Azure resource type API definitions. This feature helps provide an easier authoring experience.

Azure Quickstart Templates are Azure Resource Manager templates provided by the Azure community.

- The README.md file provides an overview of what the template does.
- The azuredeploy.json file defines the resources that will be deployed.
- The azuredeploy.parameters.json file provides the values the template needs.



Automate Azure tasks using scripts with PowerShell

How to Choose an administrative tool

Automation: Do you need to automate a set of complex or repetitive tasks? Azure PowerShell and the Azure CLI support automation, while Azure portal doesn't.

Learning curve: Do you need to complete a task quickly without learning new commands or syntax? The Azure portal doesn't require you to learn syntax or memorize commands. In Azure PowerShell and the Azure CLI, you must know the detailed syntax for each command you use.

Team skillset: Does your team have existing expertise? For example, your team may have used PowerShell to administer Windows. If so, they'll quickly become comfortable using Azure PowerShell.

Let's look at the two components that make up Azure PowerShell:

- The base PowerShell product This comes in two variants: Windows PowerShell and PowerShell 7.x, which can be installed on Windows, macOS, and Linux.
- The Azure Az PowerShell module This extra module must be installed to add the Azure-specific commands to PowerShell.

The base PowerShell product ships with cmdlets that work with features such as sessions and background jobs. You can add modules to your PowerShell installation to get cmdlets that manipulate other features. For example, there are third-party modules to work with ftp, administer your operating system, access the file system, and so on.

Cmdlets follow a verb-noun naming convention; for example, Get-Process, Format-Table, and Start-Service. There's also a convention for verb choice: "get" to retrieve data, "set" to insert or update data, "format" to format data, "out" to direct output to a destination, and so on.

Cmdlets are shipped in modules. A PowerShell Module is a DLL that includes the code to process each available cmdlet. You load cmdlets into PowerShell by loading the module in which they're contained. You can get a list of loaded modules using the Get-Module command.

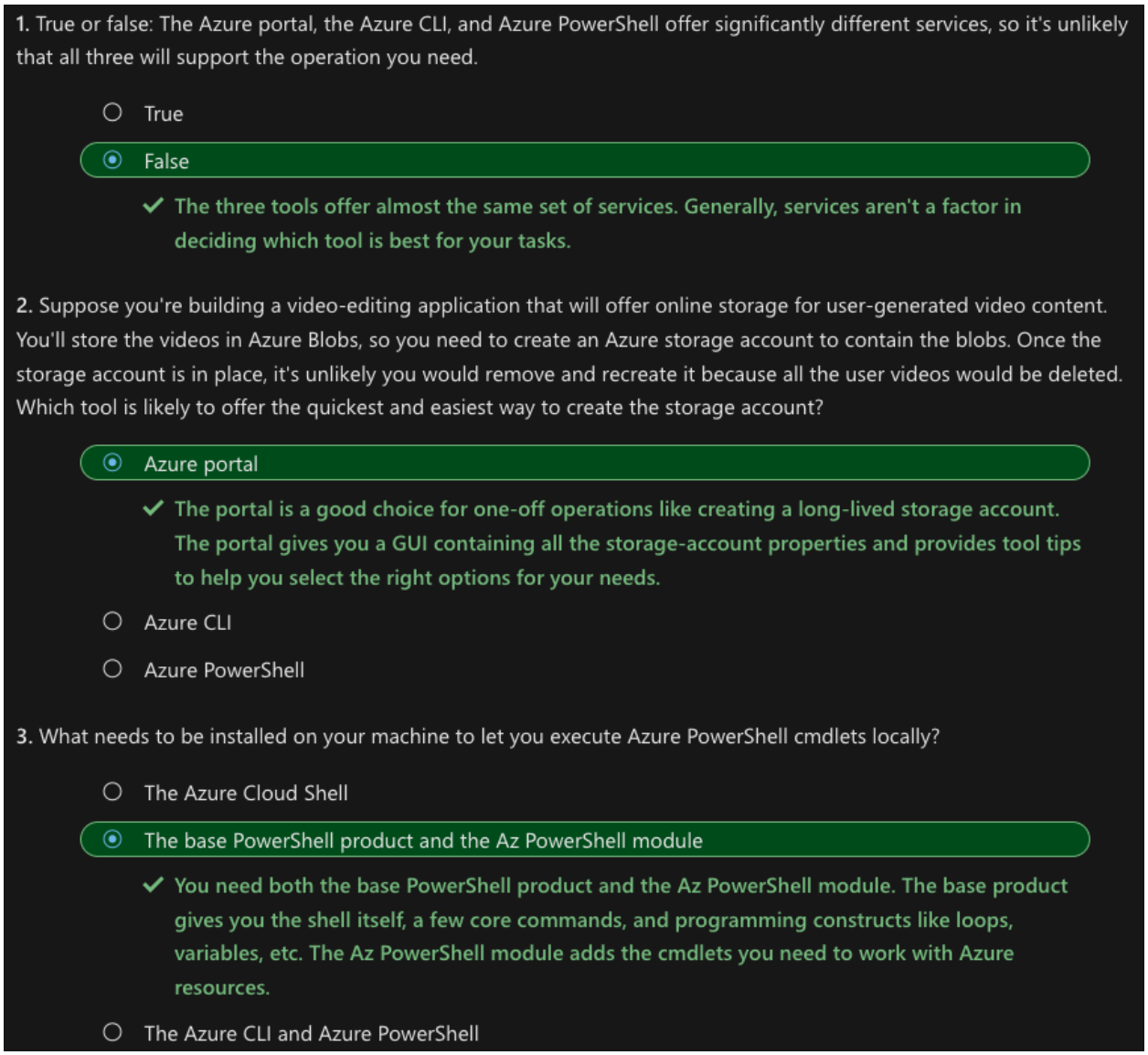
How to create a resource group with Azure PowerShell? There are four steps you need to perform:

- Import the Azure cmdlets.
- Connect to your Azure subscription.
- Create the resource group.
- Verify that creation was successful.

Beginning with PowerShell 3.0, modules are loaded automatically when you use a cmdlet within the module. It's no longer necessary to manually import PowerShell modules unless you've changed the default module autoloading settings.

A PowerShell script is a text file containing commands and control constructs. The commands are invocations of cmdlets. The control constructs are programming features like loops, variables, parameters, comments, etc., supplied by PowerShell.

PowerShell script files have a .ps1 file extension. You can create and save these files with any text editor.



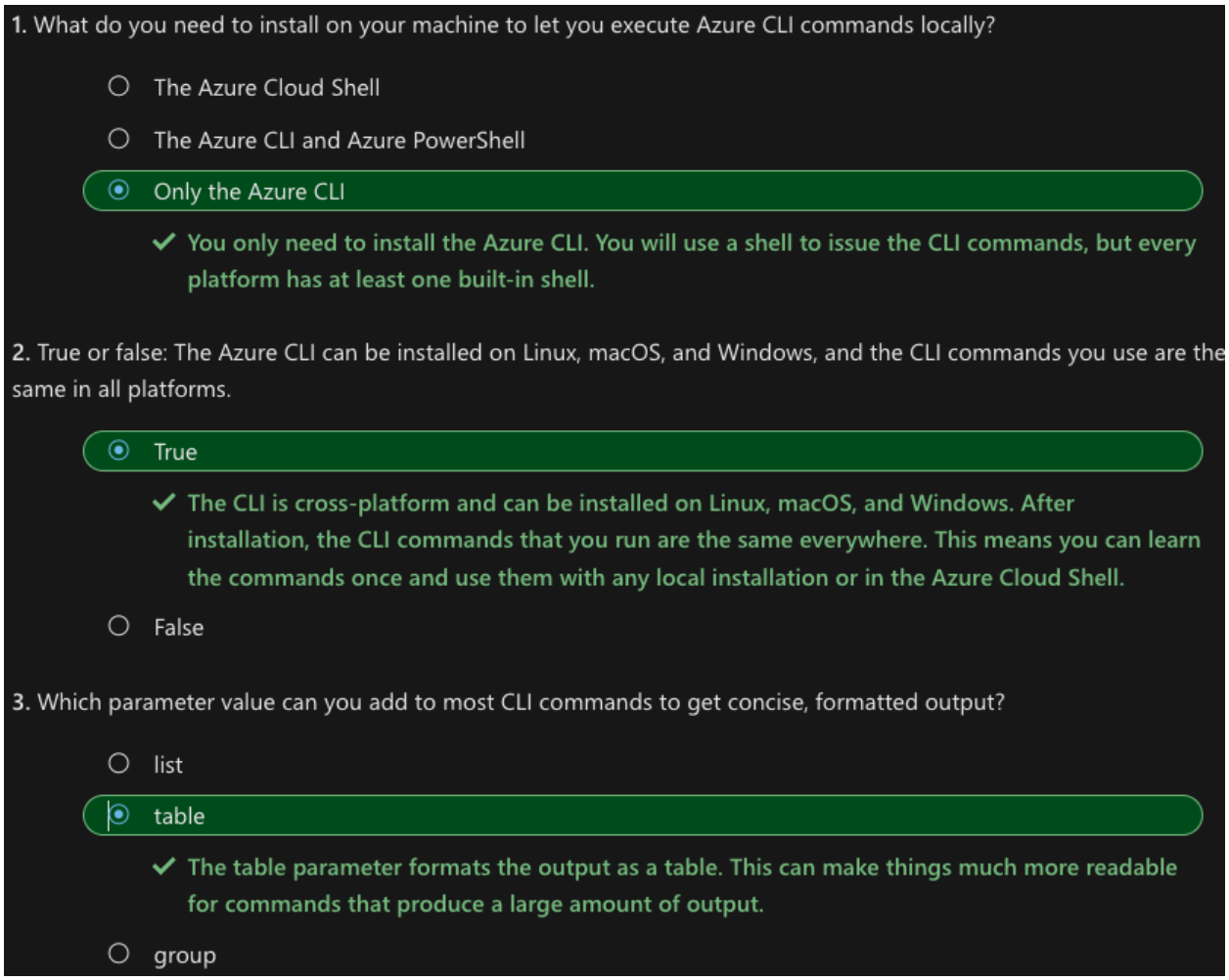
Control Azure services with the CLI

The Azure CLI provides cross-platform command-line tools for managing Azure resources, and you can easily install it locally on Linux, Mac, or Windows computers. You can also use the Azure CLI from a browser through the Azure Cloud Shell.

On both Linux and macOS, you'll use a package manager to install the Azure CLI. The recommended package manager differs by OS and distribution:

Linux: apt-get on Ubuntu, yum on Red Hat, and zypper on OpenSUSE
Mac: Homebrew
The Azure CLI is available in the Microsoft repository, so you'll first need to add that repository to your package manager.

On Windows, you can install the Azure CLI by downloading and running an MSI file.



Deploy Azure infrastructure by using JSON ARM templates

ARM templates are idempotent, which means you can deploy the same template many times and get the same resource types in the same state.

ARM templates allow you to automate deployments and use the practice of infrastructure as code (IaC).

Resource Manager also has built-in validation. It checks the template before starting the deployment to make sure the deployment will succeed.

If your deployments become more complex, you can break your ARM templates into smaller, reusable components. You can link these smaller templates together at deployment time. You can also nest templates inside other templates.

You can also integrate your ARM templates into continuous integration and continuous deployment (CI/CD) tools like Azure Pipelines, which can automate your release pipelines for fast and reliable application and infrastructure updates.

You can deploy an ARM template to Azure in one of the following ways:

- Deploy a local template.
- Deploy a linked template.
- Deploy in a continuous deployment pipeline.

In the parameters section of the template, you specify which values you can input when you deploy the resources. You're limited to 256 parameters in a template. Parameter definitions can use most template functions.

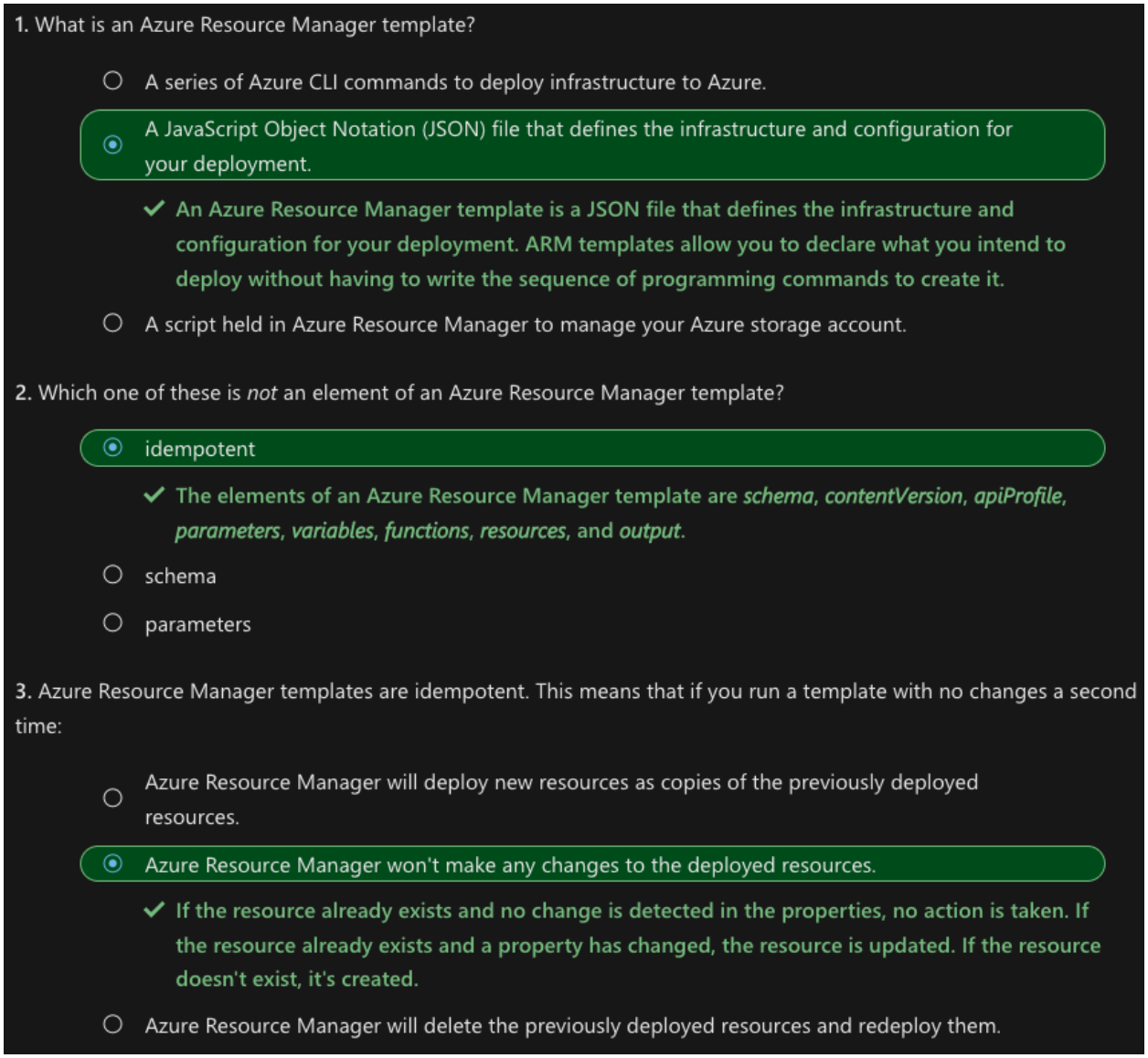
The allowed types of parameters are:

- string
- secureString
- integers
- boolean
- object
- secureObject
- array

For security reasons, never hard code or provide default values for usernames and/or passwords in templates. Always use parameters for usernames and passwords (or secrets). Use secureString for all passwords and secrets.

In the outputs section of your ARM template, you can specify values that will be returned after a successful deployment. Here are the elements that make up the outputs section.

```
"outputs": {
  "<output-name>": {
    "condition": "<boolean-value-whether-to-output-value>",
    "type": "<type-of-output-value>",
    "value": "<output-value-expression>",
    "copy": {
      "count": <number-of-iterations>,
      "input": <values-for-the-variable>
    }
  }
}
```



AZ-104: Manage identities and governance in Azure

Configure Azure Active Directory

Configure user and group accounts

Configure subscriptions

Configure Azure Policy

Configure role-based access control

Create Azure users and groups in Azure Active Directory

Secure your Azure resources with Azure role-based access control (Azure RBAC)

Allow users to reset their password with Azure Active Directory self-service password reset