# Azure Resource Manager

When a user sends a request from any of the Azure tools, APIs, or SDKs, Resource Manager receives the request. It authenticates and authorizes the request.

Resource Manager sends the request to the Azure service, which takes the requested action.

Because all requests are handled through the same API, you see consistent results and capabilities in all the different tools.



## Choose Azure Resource Manager templates

Consider the following advantages of using templates:

* Declarative syntax: Azure Resource Manager templates allow you to create and deploy an entire Azure infrastructure declaratively. For example, you can deploy not only virtual machines, but also the network infrastructure, storage systems, and any other resources you may need.
* Repeatable results: Repeatedly deploy your infrastructure throughout the development lifecycle and have confidence your resources are deployed in a consistent manner. Templates are idempotent, which means you can deploy the same template many times and get the same resource types in the same state. You can develop one template that represents the desired state, rather than developing lots of separate templates 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 template through one command, rather than through multiple imperative commands.

## Template file

Within your template, you can write template expressions that extend the capabilities of JSON. These expressions make use of the functions provided by Resource Manager.

The template has the following sections:

* [Parameters](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/parameters) - Provide values during deployment that allow the same template to be used with different environments.
* [Variables](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/variables) - Define values that are reused in your templates. They can be constructed from parameter values.
* [User-defined functions](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/user-defined-functions) - Create customized functions that simplify your template.
* [Resources](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/resource-declaration) - Specify the resources to deploy.
* [Outputs](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/outputs) - Return values from the deployed resources.

## Multi-tiered solutions

With Resource Manager, you can create a template (in JSON format) that defines the infrastructure and configuration of your Azure solution. By using a template, you can repeatedly deploy your solution throughout its lifecycle and have confidence your resources are deployed in a consistent state.

When you deploy a template, Resource Manager converts the template into REST API operations. For example, when Resource Manager receives a template with the following resource definition:

JSON:

"resources": [ {

"type": "Microsoft.Storage/storageAccounts",

"apiVersion": "2019-04-01",

"name": "mystorageaccount",

"location": "westus",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {}

} ]

It converts the definition to the following REST API operation, which is sent to the Microsoft.Storage resource provider:

HTTP:

PUT

https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Storage/storageAccounts/mystorageaccount?api-version=2019-04-01

REQUEST BODY

{

"location": "westus",

"sku": {

"name": "Standard\_LRS"

},

"kind": "StorageV2",

"properties": {}

}

*Notice that the****apiVersion****you set in the template for the resource is used as the API version for the REST operation. You can repeatedly deploy the template and have confidence it will continue to work. By using the same API version, you don't have to worry about breaking changes that might be introduced in later versions.*

### You can deploy a template using any of the following options:

* Azure portal
* Azure CLI
* PowerShell
* REST API
* Button in GitHub repository
* Azure Cloud Shell

## Defining multi-tiered templates

How you define templates and resource groups is entirely up to you and how you want to manage your solution. For example, you can deploy a three tier application through a single template to a single resource group (image-1).

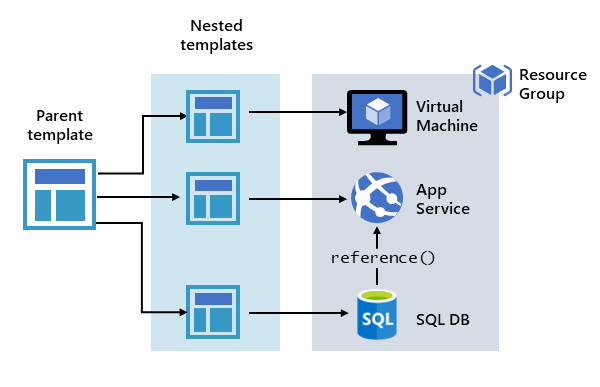
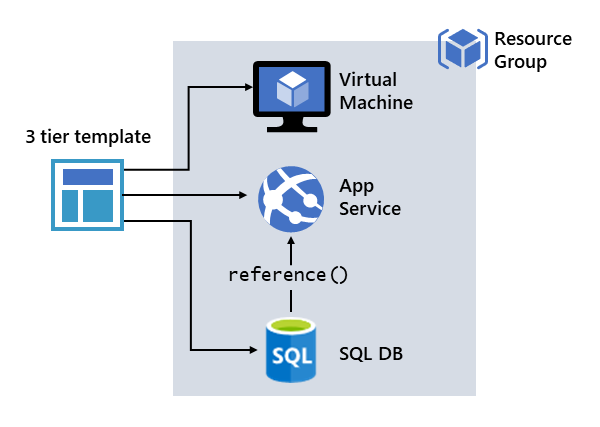


image-1 image-2

But, you don't have to define your entire infrastructure in a single template. Often, it makes sense to divide your deployment requirements into a set of targeted, purpose-specific templates. You can easily reuse these templates for different solutions.

To deploy a particular solution, you create a master template that links all the required templates. The above image-2 shows how to deploy a three-tier solution through a parent template that includes three nested templates.

**If you envision your tiers having separate lifecycles**, you can deploy your three tiers to **separate resource groups**. The resources can still be linked to resources in other resource groups.

**Azure Resource Manager analyzes dependencies to ensure resources are created in the correct order.** **If one resource relies on a value from another resource** (such as a virtual machine needing a storage account for disks), **you set a dependency**. For more information, see Defining dependencies in Azure Resource Manager templates.

You can also use the template for updates to the infrastructure. For example, you can add a resource to your solution and add configuration rules for the resources that are already deployed. If the template specifies creating a resource but that resource already exists, Azure Resource Manager performs an update instead of creating a new asset. Azure Resource Manager updates the existing asset to the same state as it would be as new.

Resource Manager provides extensions for scenarios when you need additional operations such as installing particular software that isn't included in the setup. If you're already using a configuration management service, like DSC, Chef or Puppet, you can continue working with that service by using extensions.

**Finally, the template becomes part of the source code for your app. You can check it in to your source code repository and update it as your app evolves. You can edit the template through Visual Studio.**

### Share templates

After creating your template, you may wish to share it with other users in your organization. [Template specs](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-specs) enable you to store a template as a resource type. You use role-based access control to manage access to the template spec. Users with read access to the template spec can deploy it, but not change the template.

This approach means you can safely share templates that meet your organization's standards.

## Conditional deployment

Sometimes you need to optionally deploy a resource in an Azure Resource Manager template (Azure Resource Manager template). Use the condition element to specify whether the resource is deployed. The value for the condition resolves to true or false. When the value is true, the resource is created. When the value is false, the resource isn't created. The value can only be applied to the whole resource.

### Note

Conditional deployment doesn't cascade to [**child resources**](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/child-resource-name-type). If you want to conditionally deploy a resource and its child resources, you must apply the same condition to each resource type.

### New or existing resource

You can use conditional deployment to create a new resource or use an existing one. The following example shows how to use condition to deploy a new storage account or use an existing storage account. It contains a parameter named newOrExisting which is used as a condition in the resources section.



In above JSON -> When the parameter **newOrExisting** is set to **new**, the condition evaluates to true. The storage account is deployed. However, when **newOrExisting** is set to **existing**, the condition evaluates to false and the storage account isn't deployed.

### Runtime functions

If you use a reference or list function with a resource that is conditionally deployed, the function is evaluated even if the resource isn't deployed. You get an error if the function refers to a resource that doesn't exist.

Use the if function to make sure the function is only evaluated for conditions when the resource is deployed.

You set a resource as dependent on a conditional resource exactly as you would any other resource. When a conditional resource isn't deployed, Azure Resource Manager automatically removes it from the required dependencies.

### Additional resources

* [Azure Resource Manager template functions](https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/template-functions)