# What are ARM templates?

To implement infrastructure as code for your Azure solutions, use Azure Resource Manager templates (ARM templates). The template is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project.

**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 - Provide values during deployment that allow the same template to be used with different environments.
- Variables - Define values that are reused in your templates. They can be constructed from parameter values.
- User-defined functions - Create customized functions that simplify your template.
- Resources - Specify the resources to deploy.
- Outputs - Return values from the deployed resources.

**Template deployment process**

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:

JSONCopy

```
  "resources": [
{
   "type": "Microsoft.Storage/storageAccounts",
   "apiVersion": "2022-09-01",
   "name": "mystorageaccount",
   "location": "centralus",
   "sku": {
    "name": "Standard_LRS"
   },
   "kind": "StorageV2"
  },
]
```

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

***HTTPCopy***

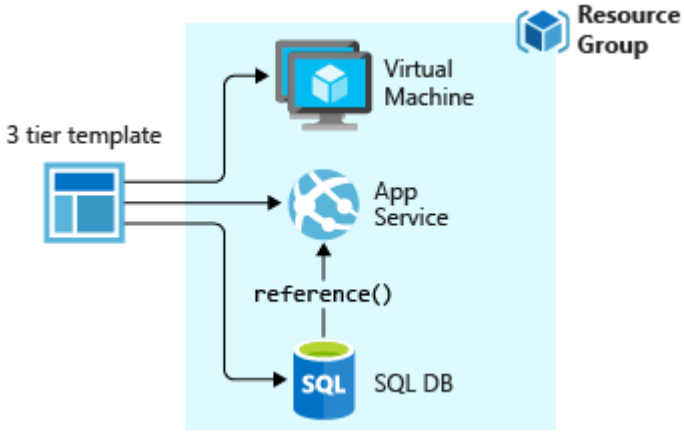
```
PUT
https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}
/providers/Microsoft.Storage/storageAccounts/mystorageaccount?api-version=2022-09-01
REQUEST BODY
{
  "location": "centralus",
  "sku": {
    "name": "Standard_LRS"
  },
```

```
  "kind": "StorageV2",
  "properties": {}
}
```

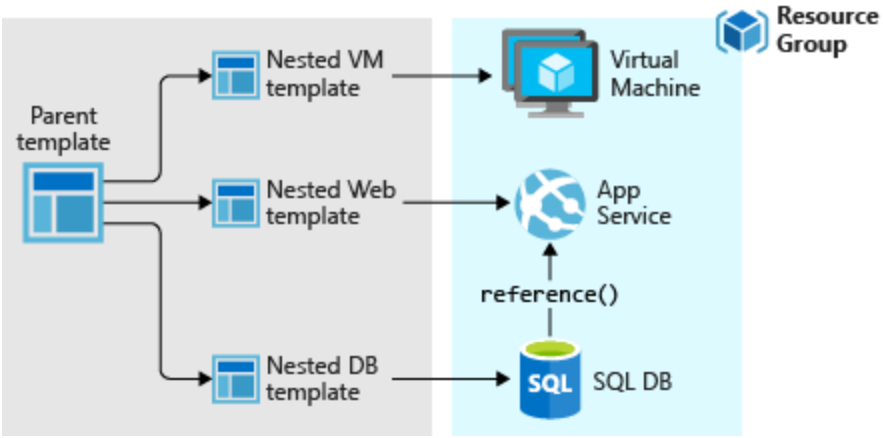
Notice that the **apiVersion** you set in the template for the resource is used as the API version for the REST operation. You can repeatedly deploy the template and have confidence it will continue to work. By using the same API version, you don't have to worry about breaking changes that might be introduced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

**Template design**

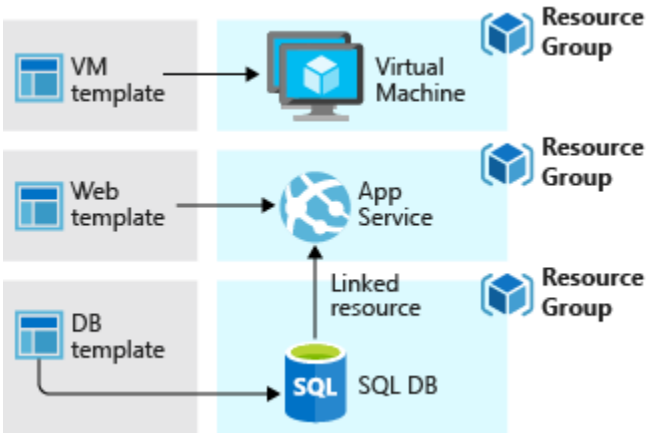
How you define templates and resource groups is entirely up to you and how you want to manage your solution. For example, you can deploy your three tier application through a single template to a single resource group.




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 main template that links all the required templates. The following image 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. Notice the resources can still be linked to resources in other resource groups.

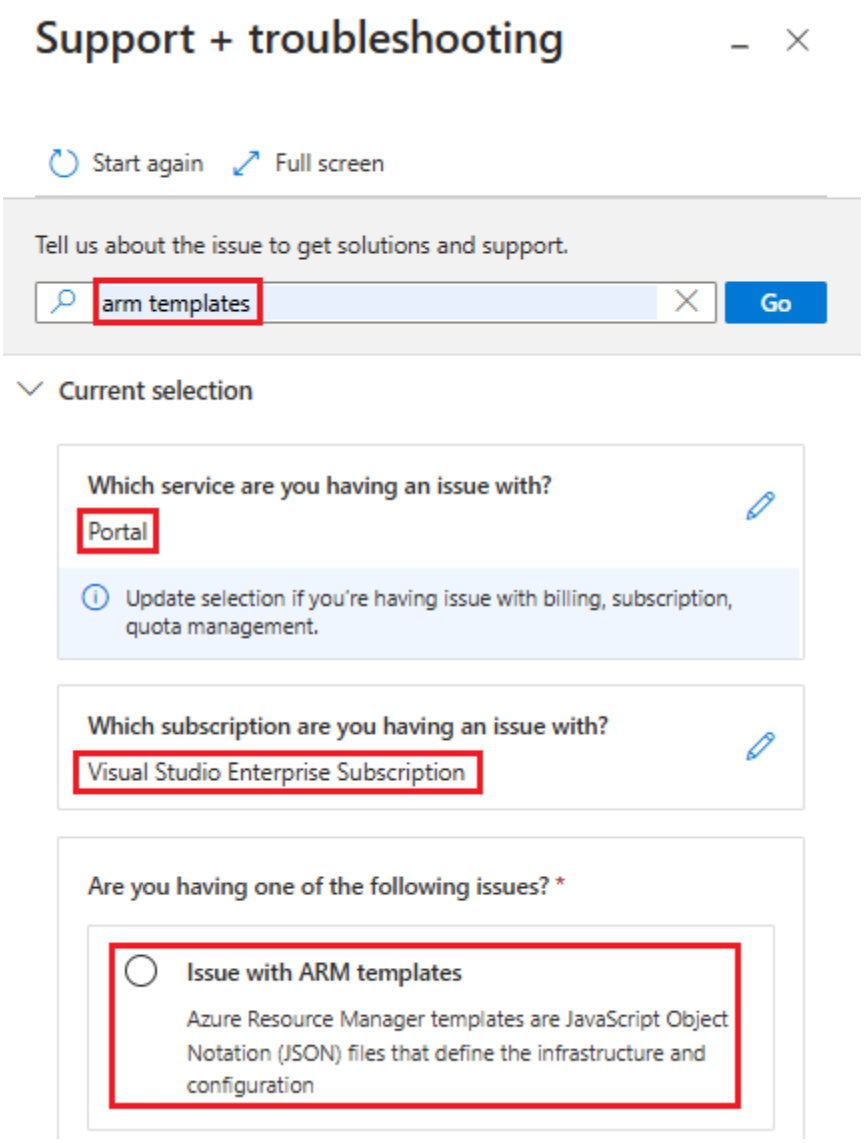



For information about nested templates, see Using linked templates with Azure Resource Manager.

## Share templates

After creating your template, you may wish to share it with other users in your organization. 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.




**Download the template using the portal**

1. Log in to the Azure portal.
2. One the left menu, select **Virtual Machines**.
3. Select the virtual machine from the list.
4. Select **Export template**.
5. Select **Download** from the menu at the top and save the .zip file to your local computer.
6. Open the .zip file and extract the files to a folder. The .zip file contains:
    - parameters.json
    - template.json

The template.json file is the template.

**Template format**

In its simplest structure, a template has the following elements:

JSONCopy

```
{
  "$schema":                    "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "languageVersion": "",
  "contentVersion": "",
  "apiProfile": "",
  "definitions": { },
  "parameters": { },
  "variables": { },
  "functions": [ ],
  "resources": [ ], /* or "resources": { } with languageVersion 2.0 */
  "outputs": { }
}
```

Expand table

| Element name | Required | Description |
|---|---|---|
| **$schema** | Yes | Location of the JavaScript Object Notation (JSON) schema file that describes the version of the template language. The version number you use depends on the scope of the deployment and your JSON editor.<br><br>If you're using Visual Studio Code with the Azure Resource Manager tools extension, use the latest version for resource group deployments:<br>https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#<br><br>Other editors (including Visual Studio) may not be able to process this schema. For those editors, use:<br>https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#<br><br>For subscription deployments, use:<br>https://schema.management.azure.com/schemas/2018-05-01/subscriptionDeploymentTemplate.json#<br><br>For management group deployments, use:<br>https://schema.management.azure.com/schemas/2019-08-01/managementGroupDeploymentTemplate.json#<br><br>For tenant deployments, use:<br>https://schema.management.azure.com/schemas/2019-08-01/tenantDeploymentTemplate.json# |
| **languageVersion** | No | Language version of the template. To view the enhancements of languageVersion 2.0, see languageVersion 2.0. |

| | | |
|---|---|---|
| **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. When deploying resources using the template, this value can be used to make sure that the right template is being used. |
| **apiProfile** | No | An API version that serves as a collection of API versions for resource types. Use this value to avoid having to specify API versions for each resource in the template. When you specify an API profile version and don't specify an API version for the resource type, Resource Manager uses the API version for that resource type that is defined in the profile.<br><br>The API profile property is especially helpful when deploying a template to different environments, such as Azure Stack and global Azure. Use the API profile version to make sure your template automatically uses versions that are supported in both environments. For a list of the current API profile versions and the resources API versions defined in the profile, see API Profile.<br><br>For more information, see Track versions using API profiles. |
| **definitions** | No | Schemas that are used to validate array and object values. Definitions are only supported in languageVersion 2.0. |
| **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 or subscription. |
| **outputs** | No | Values that are returned after deployment. |

# Set resource location in ARM template

When deploying an Azure Resource Manager template (ARM template), you must provide a location for each resource. The location doesn't need to be the same location as the resource group location.

**Get available locations**

Different resource types are supported in different locations. To get the supported locations for a resource type, use Azure PowerShell or Azure CLI.

- PowerShell
- Azure CLI

Azure PowerShellCopy

Open Cloud Shell

```
((Get-AzResourceProvider -ProviderNamespace Microsoft.Batch).ResourceTypes `
  | Where-Object ResourceTypeName -eq batchAccounts).Locations
```

## Use location parameter

To allow for flexibility when deploying your template, use a parameter to specify the location for resources. Set the default value of the parameter to **resourceGroup().location.**
The following example shows a storage account that is deployed to a location specified as a parameter:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountType": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "allowedValues": [
        "Standard_LRS",
        "Standard_GRS",
        "Standard_ZRS",
        "Premium_LRS"
      ],
      "metadata": {
        "description": "Storage Account type"
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    }
  },
  "variables": {
    "storageAccountName": "[format('storage{0}', uniqueString(resourceGroup().id))]"
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2022-09-01",
      "name": "[variables('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('storageAccountType')]"
      },
      "kind": "StorageV2",
      "properties": {}
    }
```

```
 ],
 "outputs": {
  "storageAccountName": {
   "type": "string",
   "value": "[variables('storageAccountName')]"
  }
 }
}
```

# Define the order for deploying resources in ARM templates

When deploying resources, you may need to make sure some resources exist before other resources. For example, you need a logical SQL server before deploying a database. You establish this relationship by marking one resource as dependent on the other resource. Use the dependsOn element to define an explicit dependency. Use the **reference** or **list** functions to define an implicit dependency.
Azure Resource Manager evaluates the dependencies between resources, and deploys them in their dependent order. When resources aren't dependent on each other, Resource Manager deploys them in parallel. You only need to define dependencies for resources that are deployed in the same template.

**Tip**
We recommend **Bicep** because it offers the same capabilities as ARM templates and the syntax is easier to use. To learn more, see **resource dependencies**.

**dependsOn**
Within your Azure Resource Manager template (ARM template), the dependsOn element enables you to define one resource as a dependent on one or more resources. Its value is a JavaScript Object Notation (JSON) array of strings, each of which is a resource name or ID. The array can include resources that are conditionally deployed. When a conditional resource isn't deployed, Azure Resource Manager automatically removes it from the required dependencies.
The following example shows a network interface that depends on a virtual network, network security group, and public IP address. For the full template, see the quickstart template for a Linux virtual machine.
JSONCopy

```
{
 "type": "Microsoft.Network/networkInterfaces",
 "apiVersion": "2022-07-01",
 "name": "[variables('networkInterfaceName')]",
 "location": "[parameters('location')]",
 "dependsOn": [
  "[resourceId('Microsoft.Network/networkSecurityGroups/',
parameters('networkSecurityGroupName'))]",
  "[resourceId('Microsoft.Network/virtualNetworks/', parameters('virtualNetworkName'))]",
  "[resourceId('Microsoft.Network/publicIpAddresses/', variables('publicIpAddressName'))]"
 ],
```

```
  ...
}
```