100 XP

Add flexibility to your Azure Resource Manager template by using parameters and outputs

8 minutes

In the last unit, you created an Azure Resource Manager (ARM) template and added an Azure storage account to the ARM template. You might have noticed that there's a problem with your template. The storage account name is hardcoded. You can only use this template to deploy the same storage account every time. To deploy a storage account with a different name, you would have to create a new template, which isn't a practical way to automate your deployments. The storage account SKU is also hardcoded, which means you can't vary the type of storage account for different environments. Recall that in our scenario each deployment might have a different type of storage account. You can make your template more reusable by adding a parameter for the storage account SKU.

In this unit, you learn about the *parameters* and *outputs* sections of the template.

ARM template parameters

ARM template parameters enable you to customize the deployment by providing values that are tailored for a particular environment. For example, you pass in different values based on whether you're deploying to an environment for development, test, production, or others. For example, the previous template uses the *Standard_LRS* storage account SKU. You can reuse this template for other deployments that create a storage account by making the name of the storage account SKU a parameter. Then, you pass in the name of the SKU you want for this particular deployment when the template is deployed. You can do this step either at the command line or by using a parameter file.

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 available properties for a parameter are:

JSON

```
"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>"
    }
  }
}
```

The allowed types of parameters are:

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

Recommendations for using parameters

Use parameters for settings that vary according to the environment; for example, SKU, size, or capacity. Also use parameters for resource names that you want to specify yourself for easy identification or to comply with internal naming conventions. Provide a description for each parameter, and use default values whenever possible.

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. If you pass sensitive data in a JSON object, use the secureObject type. Template parameters with *secureString* or *secureObject* types can't be read or harvested after the deployment of the resource.

Use parameters in an ARM template

In the parameters section of the ARM template, specify the parameters that can be input when you deploy the resources. You're limited to 256 parameters in a template.

Here's an example of a template file with a parameter for the storage account SKU defined in the parameters section of the template. You can provide a default for the parameter to be used if no value is specified at execution.

JSON

```json
"parameters": {
  "storageAccountType": {
    "type": "string",
    "defaultValue": "Standard_LRS",
    "allowedValues": [
      "Standard_LRS",
      "Standard_GRS",
      "Standard_ZRS",
      "Premium_LRS"
    ],
    "metadata": {
      "description": "Storage Account type"
    }
  }
}
```

Then, use the parameter in the resource definition. The syntax is `[parameters('name of the parameter')]`. You use the `parameters` function. You learn more about functions in the next module.

JSON

```json
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2019-04-01",
    "name": "learntemplatestorage123",
    "location": "[resourceGroup().location]",
    "sku": {
      "name": "[parameters('storageAccountType')]"
    },
    "kind": "StorageV2",
    "properties": {
      "supportsHttpsTrafficOnly": true
    }
  }
]
```

When you deploy the template, you can give a value for the parameter. Notice the last line in the following command:

Azure CLI

Azure CLI

```
templateFile="azuredeploy.json"
az deployment group create \
  --name testdeployment1 \
  --template-file $templateFile \
  --parameters storageAccountType=Standard_LRS
```

ARM template outputs

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.

JSON

```
"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>
    }
  }
}
```

Element	Description
output-name	Must be a valid JavaScript identifier.
condition	(Optional) A Boolean value that indicates whether this output value is returned. When true, the value is included in the output for the deployment. When false, the output value is skipped for this deployment. When not specified, the default value is true.
type	The type of the output value.
value	(Optional) A template language expression that's evaluated and returned as an output value.

Element	Description
copy	(Optional) Copy is used to return more than one value for an output.

Use outputs in an ARM template

Here's an example to output the storage account's endpoints.

JSON

```json
"outputs": {
  "storageEndpoint": {
    "type": "object",
    "value": "[reference('learntemplatestorage123').primaryEndpoints]"
  }
}
```

Notice the `reference` part of the expression. This function gets the runtime state of the storage account.

Deploy an ARM template again

Recall that ARM templates are idempotent, which means you can deploy the template to the same environment again and if nothing was changed in the template, nothing will change in the environment. If a change was made to the template, for example, you changed a parameter value, only that change will be deployed. Your template can contain all of the resources you need for your Azure solution, and you can safely execute a template again. Resources will be created only if they didn't already exist and updated only if there's a change.

Next unit: Exercise - Add parameters and outputs to your Azure Resource Manager template

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆