100 XP

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

12 minutes

Sandbox activated! Time remaining: 2 hr 18 min

You have used 1 of 10 sandboxes for today. More sandboxes will be available tomorrow.

In this exercise, you add a parameter to define the Azure storage account name during deployment. You then add a parameter to define what storage account SKU is allowed and define which one to use for this deployment. You also add usefulness to the Azure Resource Manager template (ARM template) by adding an output that can be used later in the deployment process.

Create parameters for the ARM template

Here, you make your ARM template more flexible by adding parameters that can be set at runtime. Create a parameter for the `storageName` value.

1. In the *azuredeploy.json* file in Visual Studio Code, place your cursor inside the braces in the *parameters* attribute. `"parameters":{},`

2. Select `Enter`, and then enter **par**. You see a list of related snippets. Choose **new-parameter**. It adds a generic parameter to the template. It will look like this:

JSON

```json
"parameters": {
   "parameter1": {
   "type": "string",
   "metadata": {
       "description": "description"
   }
  }
},
```

3. Change the parameter to be called **storageName**, and leave the type as a string. Add a **minLength** value of **3** and a **maxLength** value of **24**. Add a description value of **The**

name of the Azure storage resource.

4. The parameter block should look like this:

JSON

```json
"parameters": {
  "storageName": {
    "type": "string",
    "minLength": 3,
    "maxLength": 24,
    "metadata": {
      "description": "The name of the Azure storage resource"
    }
  }
},
```

5. Use the new parameter in the `resources` block in both the `name` and `displayName` values. The entire file will look like this:

JSON

```json
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageName": {
      "type": "string",
      "minLength": 3,
      "maxLength": 24,
      "metadata": {
        "description": "The name of the Azure storage resource"
      }
    }
  },
  "functions": [],
  "variables": {},
  "resources": [
    {
      "name": "[parameters('storageName')]",
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2019-06-01",
      "tags": {
        "displayName": "[parameters('storageName')]"
      },
      "location": "[resourceGroup().location]",
      "kind": "StorageV2",
      "sku": {
        "name": "Standard_LRS",
        "tier": "Standard"
      }
    }
  ],
  "outputs": {}
}
```

6. Save the file.

Deploy the parameterized ARM template

Here, you change the name of the deployment to better reflect what this deployment does and fill in a value for the new parameter.

Run the following Azure CLI commands in the terminal. This snippet is the same code you used previously, but the name of the deployment is changed. Fill in a unique name for the `storageName` parameter. Remember, this name must be unique across all of Azure. You can use the unique name you created in the last unit. In that case, Azure will update the resource instead of creating a new one.

Azure CLI

```
templateFile="azuredeploy.json"
today=$(date +"%d-%b-%Y")
DeploymentName="addnameparameter-"$today

az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile \
  --parameters storageName={your-unique-name}
```

Check your deployment

1. In your browser, go back to the Azure portal. Go to your resource group, and see that there are now **3 Succeeded** deployments. Select this link.

   Notice that all three deployments are in the list.

2. Explore the *addnameparameter* deployment as you did previously.

Add another parameter to limit allowed values

Here, you use parameters to limit the values allowed for a parameter.

1. Place your cursor after the closing brace for the `storageName` parameter. Add a comma, and select `Enter`.

2. Again, enter **par**, and select **new-parameter**.

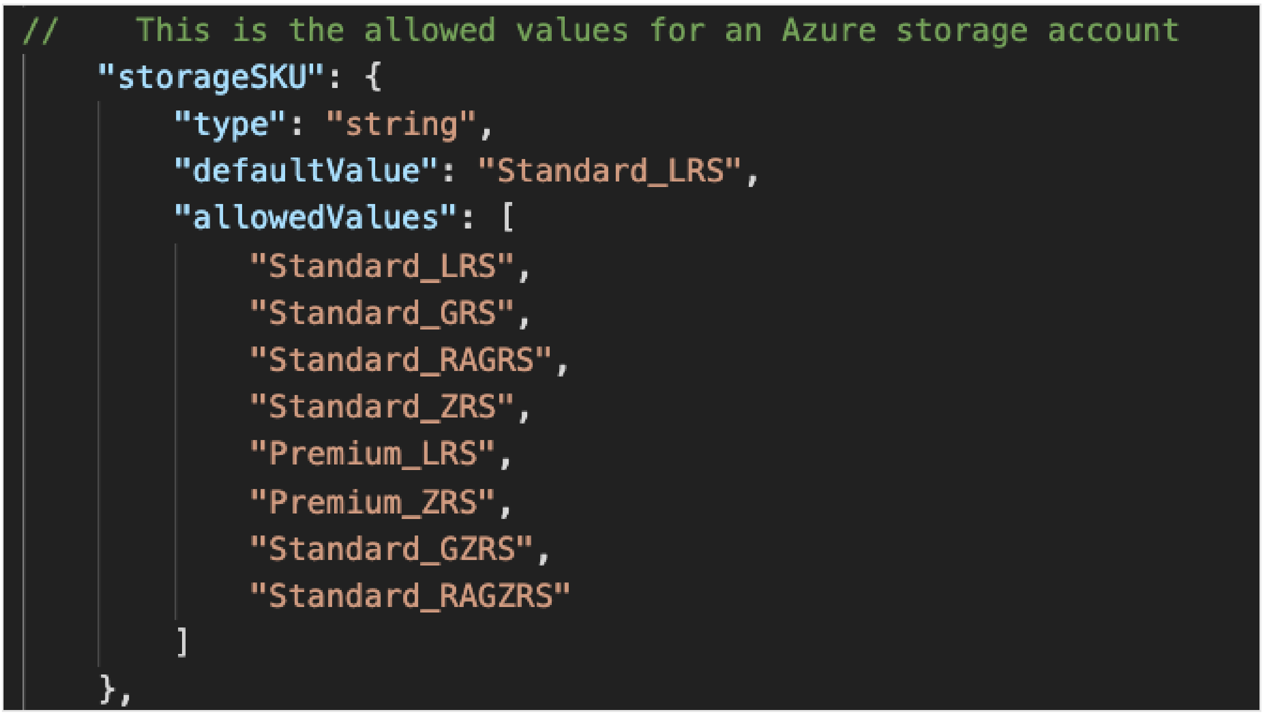
3. Change the new generic parameter to the following:

JSON

```json
"storageSKU": {
   "type": "string",
   "defaultValue": "Standard_LRS",
   "allowedValues": [
     "Standard_LRS",
     "Standard_GRS",
     "Standard_RAGRS",
     "Standard_ZRS",
     "Premium_LRS",
     "Premium_ZRS",
     "Standard_GZRS",
     "Standard_RAGZRS"
   ]
 }
```

Here, you're listing the values that this parameter will allow. If the template runs with a value that isn't allowed, the deployment will fail.

4. Add a comment to this parameter.



```
//    This is the allowed values for an Azure storage account
    "storageSKU": {
        "type": "string",
        "defaultValue": "Standard_LRS",
        "allowedValues": [
            "Standard_LRS",
            "Standard_GRS",
            "Standard_RAGRS",
            "Standard_ZRS",
            "Premium_LRS",
            "Premium_ZRS",
            "Standard_GZRS",
            "Standard_RAGZRS"
        ]
    },
```

ARM templates support `//` and `/* */` comments.

5. Update **resources** to use the `storageSKU` parameter. Take advantage of IntelliSense in Visual Studio Code to make this step easier.

JSON

```json
"sku": {
     "name": "[parameters('storageSKU')]"
   }
```

The entire file will look like this:

JSON

```json
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageName": {
      "type": "string",
      "minLength": 3,
      "maxLength": 24,
      "metadata": {
        "description": "The name of the Azure storage resource"
      }
    },
    "storageSKU": {
      "type": "string",
      "defaultValue": "Standard_LRS",
      "allowedValues": [
        "Standard_LRS",
        "Standard_GRS",
        "Standard_RAGRS",
        "Standard_ZRS",
        "Premium_LRS",
        "Premium_ZRS",
        "Standard_GZRS",
        "Standard_RAGZRS"
      ]
    }
  },
  "functions": [],
  "variables": {},
  "resources": [
    {
      "name": "[parameters('storageName')]",
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2019-06-01",
      "tags": {
        "displayName": "[parameters('storageName')]"
      },
      "location": "[resourceGroup().location]",
      "kind": "StorageV2",
      "sku": {
        "name": "[parameters('storageSKU')]",
        "tier": "Standard"
      }
    }
  ],
  "outputs": {}
}
```

6. Save the file.

Deploy the ARM template

Here, you'll deploy successfully by using a `storageSKU` parameter that's in the allowed list. Then, you'll try to deploy the template by using a `storageSKU` parameter that isn't in the allowed list. The second deployment will fail as expected.

1. Run the following commands to deploy the template. Fill in a unique name for the `storageName` parameter. Remember, this name must be unique across all of Azure. You can use the unique name you created in the last section. In that case, Azure will update the resource instead of creating a new one.

   Azure CLI

   ```
   templateFile="azuredeploy.json"
   today=$(date +"%d-%b-%Y")
   DeploymentName="addSkuParameter-"$today

   az deployment group create \
     --name $DeploymentName \
     --template-file $templateFile \
     --parameters storageSKU=Standard_GRS storageName={your-unique-name}
   ```

   Allow this deployment to finish. This deployment succeeds as expected. The allowed values prevent users of your template from passing in parameter values that don't work for the resource. Let's see what happens when you provide an invalid SKU.

2. Run the following commands to deploy the template with a parameter that isn't allowed. Here, you changed the `storageSKU` parameter to **Basic**. Fill in a unique name for the `storageName` parameter. Remember, this name must be unique across all of Azure. You can use the unique name you created in the last section. In that case, Azure will update the resource instead of creating a new one.

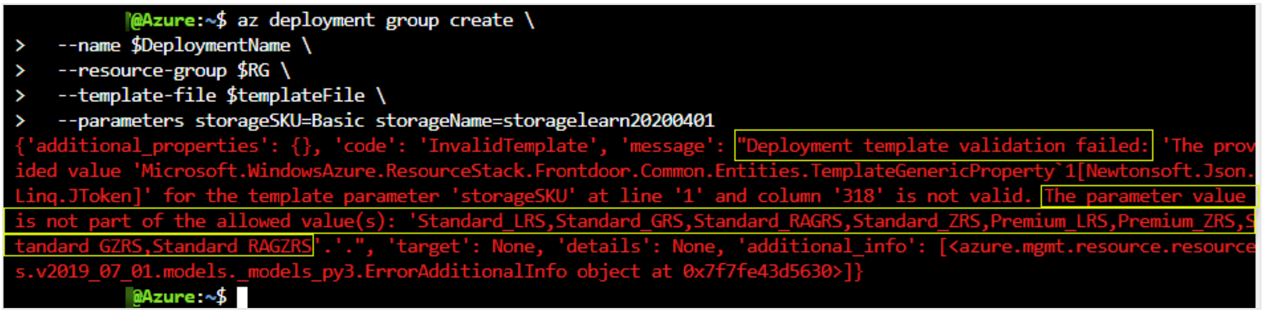
   Azure CLI

   ```
   templateFile="azuredeploy.json"
   today=$(date +"%d-%b-%Y")
   ```

```
DeploymentName="addSkuParameter-"$today

az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile \
  --parameters storageSKU=Basic storageName={your-unique-name}
```

This deployment fails. Notice the error.



```
         @Azure:~$ az deployment group create \
>   --name $DeploymentName \
>   --resource-group $RG \
>   --template-file $templateFile \
>   --parameters storageSKU=Basic storageName=storagelearn20200401
{'additional_properties': {}, 'code': 'InvalidTemplate', 'message': "Deployment template validation failed: 'The prov
ided value 'Microsoft.WindowsAzure.ResourceStack.Frontdoor.Common.Entities.TemplateGenericProperty`1[Newtonsoft.Json.
Linq.JToken]' for the template parameter 'storageSKU' at line '1' and column '318' is not valid. The parameter value
is not part of the allowed value(s): 'Standard_LRS,Standard_GRS,Standard_RAGRS,Standard_ZRS,Premium_LRS,Premium_ZRS,S
tandard_GZRS,Standard_RAGZRS'.'.", 'target': None, 'details': None, 'additional_info': [<azure.mgmt.resource.resource
s.v2019_07_01.models._models_py3.ErrorAdditionalInfo object at 0x7f7fe43d5630>]}
         @Azure:~$
```

Add output to the ARM template

Here, you add to the `outputs` section of the ARM template to output the endpoints for the storage account resource.

1. In the *azuredeploy.json* file in Visual Studio Code, place your cursor inside the braces in the outputs attribute `"outputs":{},`.

2. Select `Enter`, and then enter *out*. You see a list of related snippets. Select **new-output**. It adds a generic output to the template. It will look like this:

JSON

```json
"outputs": {
  "output1": {
    "type": "string",
    "value": "value"
  }
```

3. Change **"output1"** to **"storageEndpoint"**, then change the value of `type` to **"object"**, and finally, change the value of `value` to "**[reference(parameters('storageName')).primaryEndpoints]**". This expression is the one we described in the previous unit that gets the endpoint data. Because we specified *object* as the type, it will return the object in JSON format.

JSON

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

4. Save the file.

Deploy the ARM template with an output

Here, you deploy the template and see the endpoints output as JSON. You need to fill in a unique name for the `storageName` parameter. Remember, this name must be unique across all of Azure. You can use the unique name you created in the last section. In that case, Azure will update the resource instead of creating a new one.

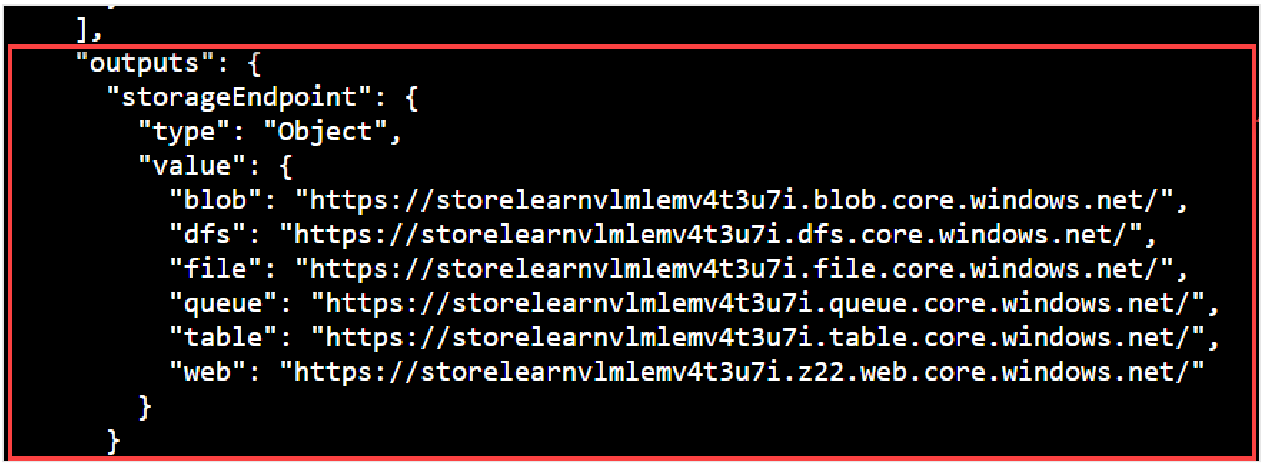
1. Run the following commands to deploy the template. Be sure to replace *{your-unique-name}* with a string unique to you.

Azure CLI

```
templateFile="azuredeploy.json"
today=$(date +"%d-%b-%Y")
DeploymentName="addoutputs-"$today

az deployment group create \
  --name $DeploymentName \
  --template-file $templateFile \
  --parameters storageSKU=Standard_LRS storageName={your-unique-name}
```

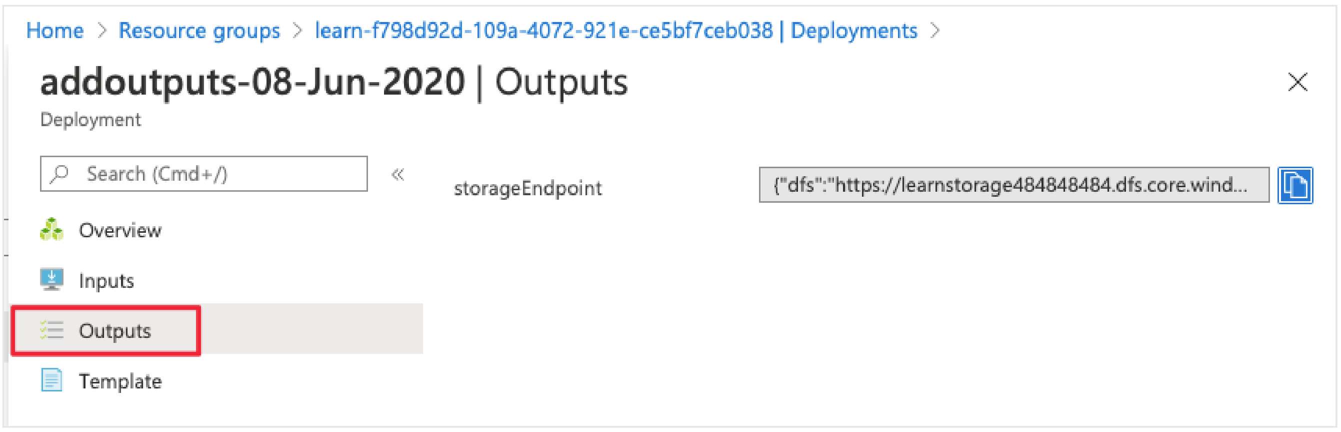
Notice the output.



```
  ],
  "outputs": {
    "storageEndpoint": {
      "type": "Object",
      "value": {
        "blob": "https://storelearnvlmlemv4t3u7i.blob.core.windows.net/",
        "dfs": "https://storelearnvlmlemv4t3u7i.dfs.core.windows.net/",
        "file": "https://storelearnvlmlemv4t3u7i.file.core.windows.net/",
        "queue": "https://storelearnvlmlemv4t3u7i.queue.core.windows.net/",
        "table": "https://storelearnvlmlemv4t3u7i.table.core.windows.net/",
        "web": "https://storelearnvlmlemv4t3u7i.z22.web.core.windows.net/"
      }
    }
```

Check your output deployment

In the Azure portal, go to your *addOutputs* deployment. You can find your output there as well.


Home > Resource groups > learn-f798d92d-109a-4072-921e-ce5bf7ceb038 | Deployments >

addoutputs-08-Jun-2020 | Outputs

Deployment

storageEndpoint {"dfs":"https://learnstorage484848484.dfs.core.wind...

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆