Use an object as a parameter in an Azure Resource Manager template

10/30/2018 • 5 minutes to read • Contributors all

In this article

When you author Azure Resource Manager templates, you can either specify resource property values directly in the template or define a parameter and provide values during deployment. It's fine to use a parameter for each property value for small deployments, but there is a limit of 255 parameters per deployment. Once you get to larger and more complex deployments you may run out of parameters.

One way to solve this problem is to use an object as a parameter instead of a value. To do this, define the parameter in your template and specify a JSON object instead of a single value during deployment. Then, reference the subproperties of the parameter using the parameter() function and dot operator in your template.

Let's take a look at an example that deploys a virtual network resource. First, let's specify a `VNetSettings` parameter in our template and set the `type` to `object`:

JSON

```
...
"parameters": {
    "VNetSettings":{"type":"object"}
},
```

Next, let's provide values for the `VNetSettings` object:

> ⓘ Note
>
> To learn how to provide parameter values during deployment, see the **parameters** section of understand the structure and syntax of Azure Resource Manager templates.

JSON

```
"parameters":{
    "VNetSettings":{
        "value":{
            "name":"VNet1",
            "addressPrefixes": [
                {
                    "name": "firstPrefix",
                    "addressPrefix": "10.0.0.0/22"
                }
            ],
            "subnets":[
                {
                    "name": "firstSubnet",
                    "addressPrefix": "10.0.0.0/24"
                },
                {
                    "name":"secondSubnet",
                    "addressPrefix":"10.0.1.0/24"
                }
            ]
        }
    }
}
```

As you can see, our single parameter actually specifies three subproperties: `name`, `addressPrefixes`, and `subnets`. Each of these subproperties either specifies a value or other subproperties. The result is that our single parameter specifies all the values necessary to deploy our virtual network.

Now let's have a look at the rest of our template to see how the `VNetSettings` object is used:

JSON

```json
...
"resources": [
    {
        "apiVersion": "2015-06-15",
        "type": "Microsoft.Network/virtualNetworks",
        "name": "[parameters('VNetSettings').name]",
        "location":"[resourceGroup().location]",
        "properties": {
          "addressSpace":{
              "addressPrefixes": [
                "[parameters('VNetSettings').addressPrefixes[0].addressPrefix]"
              ]
          },
          "subnets":[
              {
                  "name":"[parameters('VNetSettings').subnets[0].name]",
                  "properties": {
                      "addressPrefix": "[parameters('VNetSettings').subnets[0].addressPrefix]"
                  }
              },
              {
                  "name":"[parameters('VNetSettings').subnets[1].name]",
                  "properties": {
                      "addressPrefix": "[parameters('VNetSettings').subnets[1].addressPrefix]"
                  }
              }
          ]
        }
    }
  ]
```

The values of our `VNetSettings` object are applied to the properties required by our virtual network resource using the `parameters()` function with both the `[]` array indexer and the dot operator. This approach works if you just want to statically apply the values of the parameter object to the resource. However, if you want to dynamically assign an array of property values during deployment you can use a copy loop. To use a copy loop, you provide a JSON array of resource property values and the copy loop dynamically applies the values to the resource's properties.

There is one issue to be aware of if you use the dynamic approach. To demonstrate the issue, let's take a look at a typical array of property values. In this example the values for our properties are stored in a variable. Notice we have two arrays here—one named `firstProperty` and one named `secondProperty`.

JSON

```json
"variables": {
    "firstProperty": [
        {
            "name": "A",
            "type": "typeA"
        },
        {
            "name": "B",
            "type": "typeB"
        },
        {
            "name": "C",
            "type": "typeC"
        }
    ],
    "secondProperty": [
        "one","two", "three"
    ]
}
```

Now let's take a look at the way we access the properties in the variable using a copy loop.

```JSON
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    ...
    "copy": {
        "name": "copyLoop1",
        "count": "[length(variables('firstProperty'))]"
    },
    ...
    "properties": {
        "name": { "value": "[variables('firstProperty')[copyIndex()].name]" },
        "type": { "value": "[variables('firstProperty')[copyIndex()].type]" },
        "number": { "value": "[variables('secondProperty')[copyIndex()]]" }
    }
}
```

The `copyIndex()` function returns the current iteration of the copy loop, and we use that as an index into each of the two arrays simultaneously.

This works fine when the two arrays are the same length. The issue arises if you've made a mistake and the two arrays are different lengths—in this case your template will fail validation during deployment. You can avoid this issue by including all your properties in a single object, because it is much easier to see when a value is missing. For example, let's take a look another parameter object in which each element of the `propertyObject` array is the union of the `firstProperty` and `secondProperty` arrays from earlier.

```JSON
"variables": {
    "propertyObject": [
        {
            "name": "A",
            "type": "typeA",
            "number": "one"
        },
        {
            "name": "B",
            "type": "typeB",
            "number": "two"
        },
        {
            "name": "C",
            "type": "typeC"
        }
    ]
}
```

Notice the third element in the array? It's missing the `number` property, but it's much easier to notice that you've missed it when you're authoring the parameter values this way.

Using a property object in a copy loop

This approach becomes even more useful when combined with the [serial copy loop][azure-resource-manager-create-multiple], particularly for deploying child resources.

To demonstrate this, let's look at a template that deploys a network security group (NSG) with two security rules.

First, let's take a look at our parameters. When we look at our template we'll see that we've defined one parameter named `networkSecurityGroupsSettings` that includes an array named `securityRules`. This array contains two JSON objects that specify a number of settings for a security rule.

JSON

Copy

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.j-
son#",
    "contentVersion": "1.0.0.0",
    "parameters":{
      "networkSecurityGroupsSettings": {
      "value": {
          "securityRules": [
            {
              "name": "RDPAllow",
              "description": "allow RDP connections",
              "direction": "Inbound",
              "priority": 100,
              "sourceAddressPrefix": "*",
              "destinationAddressPrefix": "10.0.0.0/24",
              "sourcePortRange": "*",
              "destinationPortRange": "3389",
              "access": "Allow",
              "protocol": "Tcp"
            },
            {
              "name": "HTTPAllow",
              "description": "allow HTTP connections",
              "direction": "Inbound",
              "priority": 200,
              "sourceAddressPrefix": "*",
              "destinationAddressPrefix": "10.0.1.0/24",
              "sourcePortRange": "*",
              "destinationPortRange": "80",
              "access": "Allow",
              "protocol": "Tcp"
            }
          ]
        }
      }
    }
  }
```

Now let's take a look at our template. Our first resource named `NSG1` deploys the NSG. Our second resource named `loop-0` performs two functions: first, it `dependsOn` the NSG so its deployment doesn't begin until `NSG1` is completed, and it is the first iteration of the sequential loop. Our third resource is a nested template that deploys our security rules using an object for its parameter values as in the last example.

JSON   Copy

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
      "networkSecurityGroupsSettings": {"type":"object"}
  },
  "variables": {},
  "resources": [
    {
      "apiVersion": "2015-06-15",
      "type": "Microsoft.Network/networkSecurityGroups",
      "name": "NSG1",
      "location":"[resourceGroup().location]",
      "properties": {
          "securityRules":[]
      }
    },
    {
        "apiVersion": "2015-01-01",
        "type": "Microsoft.Resources/deployments",
        "name": "loop-0",
        "dependsOn": [
            "NSG1"
        ],
        "properties": {
            "mode":"Incremental",
            "parameters":{},
            "template": {
                "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deployment-
Template.json#",
                "contentVersion": "1.0.0.0",
                "parameters": {},
                "variables": {},
                "resources": [],
                "outputs": {}
            }
        }
    },
    {
        "apiVersion": "2015-01-01",
        "type": "Microsoft.Resources/deployments",
        "name": "[concat('loop-', copyIndex(1))]",
        "dependsOn": [
          "[concat('loop-', copyIndex())]"
        ],
        "copy": {
          "name": "iterator",
          "count": "[length(parameters('networkSecurityGroupsSettings').securityRules)]"
        },
        "properties": {
          "mode": "Incremental",
          "template": {
            "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTem-
plate.json#",
            "contentVersion": "1.0.0.0",
           "parameters": {},
            "variables": {},
            "resources": [
                {
                    "name": "[concat('NSG1/' , parameters('networkSecurityGroupsSettings').se-
curityRules[copyIndex()].name)]",
                    "type": "Microsoft.Network/networkSecurityGroups/securityRules",
                    "apiVersion": "2016-09-01",
                    "location":"[resourceGroup().location]",
                    "properties":{
                        "description": "[parameters('networkSecurityGroupsSettings').security-
Rules[copyIndex()].description]",
                        "priority":"
[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].priority]",
                        "protocol":"
[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].protocol]",
                        "sourcePortRange": "[parameters('networkSecurityGroupsSettings').secu-
rityRules[copyIndex()].sourcePortRange]",
                        "destinationPortRange": "
[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].destinationPortRange]",
                        "sourceAddressPrefix": "
[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].sourceAddressPrefix]",
                        "destinationAddressPrefix": "[parameters('networkSecurityGroupsSet-
tings').securityRules[copyIndex()].destinationAddressPrefix]",
                        "access":"
[parameters('networkSecurityGroupsSettings').securityRules[copyIndex()].access]",
                        "direction":"[parameters('networkSecurityGroupsSettings').security-
Rules[copyIndex()].direction]"
                        }
                  }
            ],
            "outputs": {}
          }
        }
    }
  ],
  "outputs": {}
}
```

Let's take a closer look at how we specify our property values in the `securityRules` child resource. All of our properties are referenced using the `parameter()` function, and then we use the dot operator to reference our `securityRules` array, indexed by the current value of the iteration. Finally, we use another dot operator to reference the name of the object.

Try the template

An example template is available on GitHub. To deploy the template, clone the repo and run the following Azure CLI commands:

bash

```bash
git clone https://github.com/mspnp/template-examples.git
cd template-examples/example3-object-param
az group create --location <location> --name <resource-group-name>
az group deployment create -g <resource-group-name> \
    --template-uri https://raw.githubusercontent.com/mspnp/template-examples/master/example3-
object-param/deploy.json \
    --parameters deploy.parameters.json
```

Next steps

- Learn how to create a template that iterates through an object array and transforms it into a JSON schema. See Implement a property transformer and collector in an Azure Resource Manager template