Implement a property transformer and collector in an Azure Resource Manager template

10/30/2018 • 6 minutes to read • Contributors

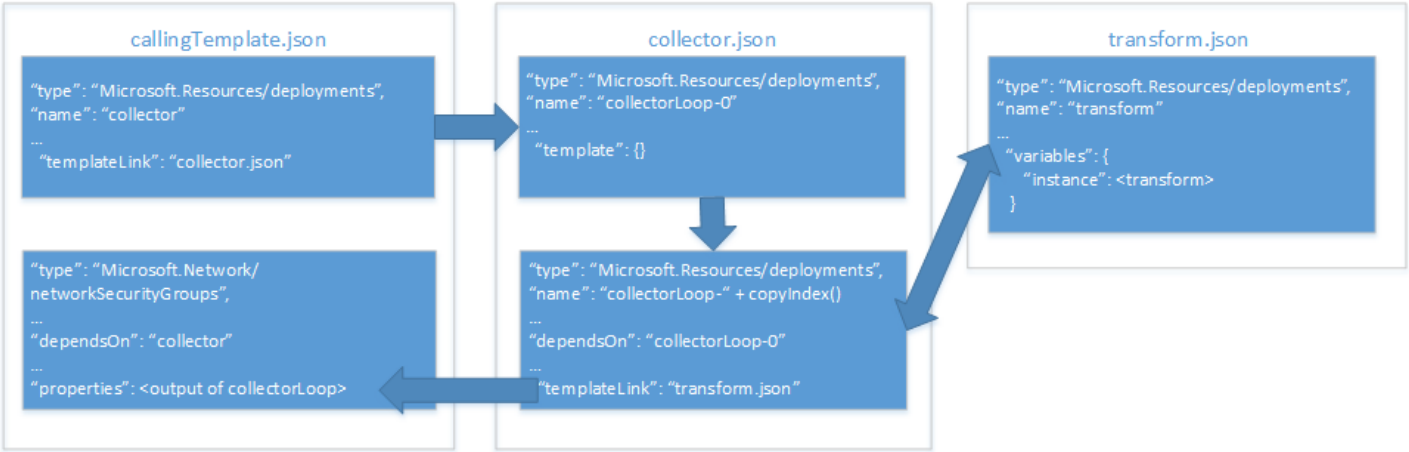
In this article

In use an object as a parameter in an Azure Resource Manager template, you learned how to store resource property values in an object and apply them to a resource during deployment. While this is a very useful way to manage your parameters, it still requires you to map the object's properties to resource properties each time you use it in your template.

To work around this, you can implement a property transform and collector template that iterates your object array and transforms it into the JSON schema expected by the resource.

> **Important**
>
> This approach requires that you have a deep understanding of Resource Manager templates and functions.

Let's take a look at how we can implement a property collector and transformer with an example that deploys a network security group. The diagram below shows the relationship between our templates and our resources within those templates:



Our **calling template** includes two resources:

- A template link that invokes our **collector template**.
- The network security group resource to deploy.

Our **collector template** includes two resources:

- An **anchor** resource.
- A template link that invokes the transform template in a copy loop.

Our **transform template** includes a single resource: an empty template with a variable that transforms our `source` JSON to the JSON schema expected by our network security group resource in the **main template**.

Parameter object

We'll be using our `securityRules` parameter object from [objects as parameters](). Our **transform template** will transform each object in the `securityRules` array into the JSON schema expected by the network security group resource in our **calling template**.

JSON Copy

```
{
    "$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"
            }
          ]
        }
      }
    }
  }
```

Let's look at our **transform template** first.

Transform template

Our **transform template** includes two parameters that are passed from the **collector template**:

- `source` is an object that receives one of the property value objects from the property array. In our example, each object from the `"securityRules"` array will be passed in one at a time.
- `state` is an array that receives the concatenated results of all the previous transforms. This is the collection of transformed JSON.

Our parameters look like this:

JSON Copy

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "source": { "type": "object" },
    "state": {
      "type": "array",
      "defaultValue": [ ]
    }
  },
```

Our template also defines a variable named `instance`. It performs the actual transform of our `source` object into the required JSON schema:

JSON Copy

```
  "variables": {
    "instance": [
      {
        "name": "[parameters('source').name]",
        "properties":{
            "description": "[parameters('source').description]",
            "protocol": "[parameters('source').protocol]",
            "sourcePortRange": "[parameters('source').sourcePortRange]",
            "destinationPortRange": "[parameters('source').destinationPortRange]",
            "sourceAddressPrefix": "[parameters('source').sourceAddressPrefix]",
            "destinationAddressPrefix": "[parameters('source').destinationAddressPrefix]",
            "access": "[parameters('source').access]",
            "priority": "[parameters('source').priority]",
            "direction": "[parameters('source').direction]"
        }
      }
    ]

  },
```

Finally, the `output` of our template concatenates the collected transforms of our `state` parameter with the current transform performed by our `instance` variable:

JSON Copy

```
    "resources": [],
    "outputs": {
    "collection": {
      "type": "array",
      "value": "[concat(parameters('state'), variables('instance'))]"
    }
```

Next, let's take a look at our **collector template** to see how it passes in our parameter values.

Collector template

Our **collector template** includes three parameters:

- `source` is our complete parameter object array. It's passed in by the **calling template**. This has the same name as the `source` parameter in our **transform template** but there is one key difference that you may have already noticed: this is the complete array, but we only pass one element of this array to the **transform template** at a time.

- `transformTemplateUri` is the URI of our **transform template**. We're defining it as a parameter here for template reusability.
- `state` is an initially empty array that we pass to our **transform template**. It stores the collection of transformed parameter objects when the copy loop is complete.

Our parameters look like this:

JSON Copy

```
  "parameters": {
    "source": { "type": "array" },
    "transformTemplateUri": { "type": "string" },
    "state": {
      "type": "array",
      "defaultValue": [ ]
    }
```

Next, we define a variable named `count`. Its value is the length of the `source` parameter object array:

JSON Copy

```
  "variables": {
    "count": "[length(parameters('source'))]"
  },
```

As you might suspect, we use it for the number of iterations in our copy loop.

Now let's take a look at our resources. We define two resources:

- `loop-0` is the zero-based resource for our copy loop.
- `loop-` is concatenated with the result of the `copyIndex(1)` function to generate a unique iteration-based name for our resource, starting with `1`.

Our resources look like this:

JSON Copy

```
  "resources": [
    {
      "type": "Microsoft.Resources/deployments",
      "apiVersion": "2015-01-01",
      "name": "loop-0",
      "properties": {
        "mode": "Incremental",
        "parameters": { },
        "template": {
          "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTem-
plate.json#",
          "contentVersion": "1.0.0.0",
          "parameters": { },
          "variables": { },
          "resources": [ ],
          "outputs": {
            "collection": {
              "type": "array",
              "value": "[parameters('state')]"
            }
          }
        }
      }
    },
    {
      "type": "Microsoft.Resources/deployments",
```

```
      "apiVersion": "2015-01-01",
      "name": "[concat('loop-', copyindex(1))]",
      "copy": {
        "name": "iterator",
        "count": "[variables('count')]",
        "mode": "serial"
      },
      "dependsOn": [
        "loop-0"
      ],
      "properties": {
        "mode": "Incremental",
        "templateLink": { "uri": "[parameters('transformTemplateUri')]" },
        "parameters": {
          "source": { "value": "[parameters('source')[copyindex()]]" },
          "state": { "value": "[reference(concat('loop-', copyindex())).outputs.collection.val-
ue]" }
        }
      }
    }
  ],
```

Let's take a closer look at the parameters we're passing to our **transform template** in the nested template. Recall from earlier that our `source` parameter passes the current object in the `source` parameter object array. The `state` parameter is where the collection happens, because it takes the output of the previous iteration of our copy loop—notice that the `reference()` function uses the `copyIndex()` function with no parameter to reference the `name` of our previous linked template object—and passes it to the current iteration.

Finally, the `output` of our template returns the `output` of the last iteration of our **transform template**:

JSON Copy

```
  "outputs": {
    "result": {
      "type": "array",
      "value": "[reference(concat('loop-', variables('count'))).outputs.collection.value]"
    }
  }
```

It may seem counterintuitive to return the `output` of the last iteration of our **transform template** to our **calling template** because it appeared we were storing it in our `source` parameter. However, remember that it's the last iteration of our **transform template** that holds the complete array of transformed property objects, and that's what we want to return.

Finally, let's take a look at how to call the **collector template** from our **calling template**.

Calling template

Our **calling template** defines a single parameter named `networkSecurityGroupsSettings`:

JSON Copy

```
...
"parameters": {
    "networkSecurityGroupsSettings": {
        "type": "object"
    }
```

Next, our template defines a single variable named `collectorTemplateUri`:

JSON Copy

```
"variables": {
    "collectorTemplateUri": "[uri(deployment().properties.templateLink.uri, 'collector.tem-
plate.json')]"
  }
```

As you would expect, this is the URI for the **collector template** that will be used by our linked template resource:

JSON Copy

```
{
    "apiVersion": "2015-01-01",
    "name": "collector",
    "type": "Microsoft.Resources/deployments",
    "properties": {
        "mode": "Incremental",
        "templateLink": {
            "uri": "[variables('collectorTemplateUri')]",
            "contentVersion": "1.0.0.0"
        },
        "parameters": {
            "source" : {"value": "
[parameters('networkSecurityGroupsSettings').securityRules]"},
            "transformTemplateUri": { "value": "[uri(deployment().properties.templateLink.uri,
'transform.json')]"}
        }
    }
}
```

We pass two parameters to the **collector template**:

- `source` is our property object array. In our example, it's our `networkSecurityGroupsSettings` parameter.
- `transformTemplateUri` is the variable we just defined with the URI of our **collector template**.

Finally, our `Microsoft.Network/networkSecurityGroups` resource directly assigns the `output` of the `collector` linked template resource to its `securityRules` property:

JSON Copy

```
    {
      "apiVersion": "2015-06-15",
      "type": "Microsoft.Network/networkSecurityGroups",
      "name": "networkSecurityGroup1",
      "location": "[resourceGroup().location]",
      "properties": {
        "securityRules": "[reference('collector').outputs.result.value]"
      }
    }
  ],
  "outputs": {
      "instance":{
          "type": "array",
          "value": "[reference('collector').outputs.result.value]"
      }

  }
```

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 Copy

```bash
git clone https://github.com/mspnp/template-examples.git
cd template-examples/example4-collector
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/example4-collector/deploy.json \
    --parameters deploy.parameters.json
```