Cheatography

Structure and Syntax of ARM Templates Cheat Sheet

by Ilham (ilperdan0) via cheatography.com/126607/cs/26286/

Template format

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

A template's elements, in its simplest structure. Each element has properties we can set.

Template Format Elements

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 depends on the scope of the deployment and the JSON editor.
contentVersion	Yes	Version of the template (such as 1.0.0.0). Can be any value. Used to document significant changes in 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.
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.

Template Format Elements (cont)

resources	Yes	Resource types that are deployed or updated in a resource group or subscription.
outputs	No	Values that are returned after deployment.

Parameters

```
"parameters": {
  "<parameter-name>" : {
    "type" : "<type-of-parameter-value>",
    "defaultValue": "<default-value-of-parameter-value>",
    "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>"
    }
  }
}
```

Specify which values one can input when deploying the resources. It is limited to 256 parameters in a template. Use objects that contain multiple properties to reduce the number of parameters.

Parameter Elements

Element name	Required	Description
parameter-name	Yes	Name of the parameter. Must be a valid JavaScript identifier.
type	Yes	Type of the parameter value. The allowed types and values are string, securestring, int, bool, object, secureObject, and array.
defaultValue	No	Default value for the parameter, if no value is provided for the parameter.
allowedValues	No	Array of allowed values for the parameter to make sure that the right value is provided.
minValue	No	The minimum value for int type parameters, this value is inclusive.

Parameter Elements (cont)

maxValue	No	The maximum value for int type parameters, this value is inclusive.
minLength	No	The minimum length for string, secure string, and array type parameters, this value is inclusive.
maxLength	No	The maximum length for string, secure string, and array type parameters, this value is inclusive.
description	No	Description of the parameter that is displayed to users through the portal. For more information

Define Parameters Example

```
"parameters": {
  "storageSKU": {
    "type": "string",
    "allowedValues": [
      "Standard_LRS",
      "Standard_ZRS",
      "Standard_GRS",
      "Standard_RAGRS",
      "Premium_LRS"
    ],
    "defaultValue": "Standard_LRS",
    "metadata": {
      "description": "The type of replication to
use for the storage account."
    }
  }
}
```

The above example shows a simple parameter definition. It defines a parameter named ***storageSKU***. The parameter is a string value, and only accepts values that are valid for its intended use. The parameter uses a default value when no value is provided during deployment.

Resources

```
"resources": [
  {
      "condition": "<true-to-deploy-this-resour-
ce>",
      "type": "<resource-provider-namespace/reso-
urce-type-name>",
      "apiVersion": "<api-version-of-resource>",
      "name": "<name-of-the-resource>",
      "comments": "<your-reference-notes>",
```

Resources (cont)

```
      "location": "<location-of-resource>",
      "dependsOn": [
          "<array-of-related-resource-names>"
      ],
      "tags": {
          "<tag-name1>": "<tag-value1>",
          "<tag-name2>": "<tag-value2>"
      },
      "sku": {
          "name": "<sku-name>",
          "tier": "<sku-tier>",
          "size": "<sku-size>",
          "family": "<sku-family>",
          "capacity": <sku-capacity>
      },
      "kind": "<type-of-resource>",
      "copy": {
          "name": "<name-of-copy-loop>",
          "count": <number-of-iterations>,
          "mode": "<serial-or-parallel>",
          "batchSize": <number-to-deploy-serially>
      },
      "plan": {
          "name": "<plan-name>",
          "promotionCode": "<plan-promotion-co-
de>",
          "publisher": "<plan-publisher>",
          "product": "<plan-product>",
          "version": "<plan-version>"
      },
      "properties": {
          "<settings-for-the-resource>",
          "copy": [
              {
                  "name": ,
                  "count": ,
                  "input": {}
              }
          ]
      },
      "resources": [
          "<array-of-child-resources>"
```

Resources (cont)

```
        ]
    }
]
```

To define the resources that are deployed or updated.

Resources Elements Format

Element name	Required	Description
condition	No	Boolean value that indicates whether the resource will be provisioned during this deployment. When true, the resource is created during deployment. When false, the resource is skipped for this deployment.
type	Yes	Type of the resource. This value is a combination of the namespace of the resource provider and the resource type (such as **Microsoft.Storage/storageAccounts**).
apiVersion	Yes	Version of the REST API to use for creating the resource. When creating a new template, set this value to the latest version of the resource you're deploying.
name	Yes	Name of the resource. The name must follow URI component restrictions defined in RFC3986.
comments	No	Your notes for documenting the resources in your template.
location	Varies	Supported geo-locations of the provided resource. You can select any of the available locations, but typically it makes sense to pick one that is close to your users.
dependsOn	No	Resources that must be deployed before this resource is deployed. Resource Manager evaluates the dependencies between resources and deploys them in the correct order.

Resources Elements Format (cont)

tags	No	Tags that are associated with the resource. Apply tags to logically organize resources across your subscription.
sku	No	Some resources allow values that define the SKU to deploy. For example, you can specify the type of redundancy for a storage account.
kind	No	Some resources allow a value that defines the type of resource you deploy.
copy	No	If more than one instance is needed, the number of resources to create. The default mode is parallel. Specify serial mode when you don't want all or the resources to deploy at the same time.
plan	No	Some resources allow values that define the plan to deploy. For example, you can specify the marketplace image for a virtual machine.
properties	No	Resource-specific configuration settings. The values for the properties are the same as the values you provide in the request body for the REST API operation (PUT method) to create the resource. You can also specify a copy array to create several instances of a property.
resources	No	Child resources that depend on the resource being defined. Only provide resource types that are permitted by the schema of the parent resource. Dependency on the parent resource isn't implied. You must explicitly define that dependency.

Cheatography

Structure and Syntax of ARM Templates Cheat Sheet

by Ilham (ilperdan0) via cheatography.com/126607/cs/26286/

Comments

```
{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2018-10-01",
  "name": "[variables('vmName')]", // to customize name, change it in variables
  "location": "[parameters('location')]", //defaults to resource group location
  "dependsOn": [ /  storage account and network interface must be deployed first /
    "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
    "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
  ],
```

For inline comments, you can use either // or / ... / but this syntax doesn't work with all tools. If you add this style of comment, be sure the tools you use support inline JSON comments.

Metadata

Metadata (cont)

```
      "Environment": "[parameters('environment')]"
    },
    "sku": {
      "name": "Standard_LRS"
    },
    "kind": "Storage",
    "properties": {}
  }
]
```

For outputs, add a metadata object to the output value.

```
"outputs": {
  "hostname": {
    "type": "string",
    "value": "[reference(variables('publicIPAddressName')).dnsSettings.fqdn]",
    "metadata": {
      "comments": "Return the fully qualified domain name"
    }
  },
```

You can add a metadata object almost anywhere in your template. Resource Manager ignores the object, but your JSON editor may warn you that the property isn't valid. In the object, define the properties you need.
You can't add a metadata object to user-defined functions.

Data Types

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "stringParameter": {
      "type": "string",
      "defaultValue": "option 1"
    },
    "intParameter": {
      "type": "int",
      "defaultValue": 1
    },
    "boolParameter": {
        "type": "bool",
        "defaultValue": true
    },
    "objectParameter": {
      "type": "object",
```

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "metadata": {
    "comments": "This template was developed for demonstration purposes.",
    "author": "Example Name"
  },
```

For parameters, add a metadata object with a description property.

```
"parameters": {
  "adminUsername": {
    "type": "string",
    "metadata": {
      "description": "User name for the Virtual Machine."
    }
  },
```

The following example shows both a comments element and a metadata object for Resources

```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts",
    "apiVersion": "2018-07-01",
    "name": "[concat('storage', uniqueString(resourceGroup().id))]",
    "comments": "Storage account used to store VM disks",
    "location": "[parameters('location')]",
    "metadata": {
      "comments": "These tags are needed for policy compliance."
    },
    "tags": {
      "Dept": "[parameters('deptName')]",
```

Data Types (cont)

```
      "defaultValue": {
        "one": "a",
        "two": "b"
      }
    },
    "arrayParameter": {
      "type": "array",
      "defaultValue": [ 1, 2, 3 ]
    }
  },
  "resources": [],
  "outputs": {}
}
```

Data Types Explanation

Data Types within an ARM template:
- string
- securestring
- int
- bool
- object
- secureObject
- array

Secure string uses the same format as **string**, and **secure object** uses the same format as **object**.

A parameter type as a secure string or secure object, the value of the parameter isn't saved to the deployment history and isn't logged.

Use **secure strings** for passwords and secrets.

If you set that secure value to a property that isn't expecting a secure value, the value isn't protected.

For example, if you set a secure string to a tag, that value is stored as plain text.

For integers passed as inline parameters, the range of values may be limited by the SDK or command-line tool used.

To avoid this limitation, specify large integer values in a parameter file.

Resource types apply their own limits for integer properties.

For **boolean** and **integer** values in the template, start and end string values with double quotation marks ("string value").

Objects start with a left brace ({) and end with a right brace (}).

Arrays start with a left bracket ([) and end with a right bracket (]).

Variables

```
"variables": {
  "<variable-name>": "<variable-value>",
  "<variable-name>": {
    <variable-complex-type-value>
  },
  "<variable-object-name>": {
    "copy": [
      {
        "name": "<name-of-array-property>",
        "count": <number-of-iterations>,
        "input": <object-or-value-to-repeat>
      }
    ]
  },
  "copy": [
    {
      "name": "<variable-array-name>",
      "count": <number-of-iterations>,
      "input": <object-or-value-to-repeat>
    }
  ]
}
```

In the variables section, you construct values that can be used throughout the template. It's not necessary to define variables, but they often simplify the template by reducing complex expressions. The format of each variable matches one of the data types.

Variables Example

Define variable

```
"variables": {
"storageName": "[concat(toLower(parameters('stora-
geNamePrefix')),uniqueString(resourceGroup().i-
d))]"
},
```

Use variable

```
"resources": [
{
"type": "Microsoft.Storage/storageAccounts",
"name": "[variables('storageName')]",
...
}
]
```

Functions

To define complicated expressions that you don't want to repeat throughout your template. You create the user-defined functions from expressions and functions that are supported in templates.
Function restrictions:
- The function can't access variables.
- The function can only use parameters that are defined in the function. When you use the parameters function within a - user-defined function, you're restricted to the parameters for that function.
- The function can't call other user-defined functions.
- The function can't use the reference function.
- Parameters for the function can't have default values.

Functions Format

```
"functions": [
  {
    "namespace": "<namespace-for-functions>",
    "members": {
      "<function-name>": {
        "parameters": [
          {
            "name": "<parameter-name>",
            "type": "<type-of-parameter-value>"
          }
        ],
        "output": {
          "type": "<type-of-output-value>",
          "value": "<function-return-value>"
        }
      }
    }
  }
],
```

Functions Format Elements

Element name	Required	Description
namespace	Yes	Namespace for the custom functions. Use to avoid naming conflicts with template functions.

Functions Format Elements (cont)

function-name	Yes	Name of the custom function. When calling the function, combine the function name with the namespace. For example, to call a function named `uniqueName` in the namespace contoso, use `"[contoso.uniqueName()]"`.
parameter-name	No	Name of the parameter to be used within the custom function.
parameter-value	No	Type of the parameter value. The allowed types and values are **string**, **securestring**, **int**, **bool**, **object**, **secureObject**, and **array**.
output-type	Yes	Type of the output value. Output values support the same types as function input parameters.
output-value	Yes	Template language expression that is evaluated and returned from the function.

Outputs

```
"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>
    }
  }
}
```

To specify values that are returned from deployment. Typically, it returns values from resources that were deployed.

Outputs Element Format

Element name	Required	Description
output-name	Yes	Name of the output value. Must be a valid JavaScript identifier.

Outputs Element Format (cont)

condition	No	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	Yes	Type of the output value. Output values support the same types as template input parameters. If you specify securestring for the output type, the value isn't displayed in the deployment history and can't be retrieved from another template. To use a secret value in more than one template, store the secret in a Key Vault and reference the secret in the parameter file.
value	No	Template language expression that is evaluated and returned as output value. Specify either value or copy.
copy	No	Used to return more than one value for an output. Specify value or copy.

Outputs in ARM templates

Define output values

The example shows how to return the resource ID for a public IP address:

```
"outputs": {
  "resourceID": {
    "type": "string",
    "value": "[resourceId('Microsoft.Network/publicIPAddresses', parameters('publicIPAddresses_name'))]"
  }
}
```

Get output values

PowerShell

```
(Get-AzResourceGroupDeployment `
  -ResourceGroupName <resource-group-name> `
  -Name <deployment-name>).Outputs.resourceID.value
```

Azure CLI

```
az deployment group show \
  -g <resource-group-name> \
  -n <deployment-name> \
  --query properties.outputs.resourceID.value
```

Multi-line strings

```
{
  "type": "Microsoft.Compute/virtualMachines",
  "apiVersion": "2018-10-01",
  "name": "[variables('vmName')]", // to customize name, change it in variables
  "location": "[
    parameters('location')
    ]", //defaults to resource group location
  /*
    storage account and network interface
    must be deployed first
  */
  "dependsOn": [
    "[resourceId('Microsoft.Storage/storageAccounts/', variables('storageAccountName'))]",
    "[resourceId('Microsoft.Network/networkInterfaces/', variables('nicName'))]"
  ],
```

You can break a string into multiple lines. For example, see the location property and one of the comments in the following JSON example.