



Azure ARM Templates (Azure Resource Manager Templates)

◆ 1. What is an ARM Template?

- ARM = Azure Resource Manager.
- ARM Templates are JSON files that **define the infrastructure and configuration** of Azure resources.
- They enable **Infrastructure as Code (IaC)**:
 - You **declare** what resources you want (VMs, Storage, Networks, etc.).
 - Azure **provisions them automatically**.

👉 Think of it as a **blueprint** for your Azure environment.

◆ 2. Why Use ARM Templates?

✨ Benefits:

- **Consistency** → Same configuration every time (no manual errors).
- **Automation** → Deploy resources repeatedly and reliably.
- **Declarative** → Describe the "what", not the "how".

- **Idempotent** → Running the same template again won't break resources.
 - **Scalable** → Manage single resource or full environments.
 - **CI/CD Ready** → Works with Azure DevOps, GitHub Actions, Jenkins, etc.
-

◆ 3. Structure of an ARM Template

An ARM Template is a **JSON file** with sections:

```
{  
  "$schema":  
    "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  
  "contentVersion": "1.0.0.0",  
  "parameters": {  
    "storageName": {  
      "type": "string"  
    }  
  },  
  "variables": {  
    "location": "eastus"  
  },  
  "resources": [  
    {  
      "type": "Microsoft.Storage/storageAccounts",  
      "apiVersion": "2023-01-01",  
      "name": "[parameters('storageName')]",  
      "location": "[variables('location')]",  
      "sku": { "name": "Standard_LRS" },  
      "kind": "StorageV2",  
      "properties": {}  
    }  
  "outputs": {  
    "storageEndpoint": {
```

```
        "type": "string",
        "value": "[reference(parameters('storageName')).primaryEndpoints.blob]"
    }
}
}
```

◆ 4. ARM Template Sections Explained

1. **\$schema** → Points to ARM Template schema (ensures validation).
 2. **contentVersion** → Version of the template (**1.0.0.0** by default).
 3. **parameters** → User input values (e.g., name, size, location).
 4. **variables** → Simplify template logic (e.g., reuse values).
 5. **resources** → Actual Azure resources (VMs, Storage, Networks).
 6. **outputs** → Return values after deployment (e.g., endpoints, IPs).
-

◆ 5. Deployment Methods

You can deploy ARM Templates using:



```
az deployment group create \
--resource-group MyRG \
--template-file template.json \
--parameters storageName=mystorage123
```



```
New-AzResourceGroupDeployment ` 
-ResourceGroupName MyRG ` 
-TemplateFile template.json ` 
-storageName mystorage123
```

Azure Portal

- Go to **Deploy a custom template** → Upload `template.json`.

CI/CD

- Use Azure DevOps pipelines or GitHub Actions to deploy templates automatically.

◆ 6. Parameters and Variables

- **Parameters:** External values (customizable at runtime).
- **Variables:** Internal shortcuts (computed once, reused).

Example:

```
"parameters": { 
  "vmSize": { 
    "type": "string", 
    "defaultValue": "Standard_B1s", 
    "allowedValues": [ "Standard_B1s", "Standard_B2s" ] 
  } 
}
```

◆ 7. Functions in ARM Templates

ARM supports **built-in functions** to make templates dynamic:

- `concat()` → Combine strings.
- `resourceGroup().location` → Use RG location.
- `reference()` → Get resource runtime values.
- `uniqueString()` → Generate unique names.

Example:

```
"name": "[concat('storage', uniqueString(resourceGroup().id))]"
```

◆ 8. Linked and Nested Templates

- **Nested Templates:** Include a template inside another for modularity.
- **Linked Templates:** Call external templates stored in GitHub, Storage, etc.

👉 This helps manage **large infrastructures** (microservices, multi-tier apps).

◆ 9. Outputs

Useful for returning info after deployment:

```
"outputs": {  
    "publicIP": {  
        "type": "string",  
        "value": "[reference('myPublicIP').ipAddress]"  
    }  
}
```

◆ 10. Example: VM Deployment Template

```
{  
  "$schema":  
    "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  
  "contentVersion": "1.0.0.0",  
  "parameters": {  
    "adminUsername": { "type": "string" },  
    "adminPassword": { "type": "securestring" }  
},  
  "resources": [  
    {  
      "type": "Microsoft.Compute/virtualMachines",  
      "apiVersion": "2023-03-01",  

```

```
        "id": "[resourceId('Microsoft.Network/networkInterfaces', 'myVMNic')]"  
    }  
]  
}  
}  
}  
]  
}
```

◆ 11. ARM Templates vs Bicep

- ARM Templates → JSON (harder to read/write).
- Bicep → New DSL for ARM (simpler syntax, auto-compiles to ARM JSON).

👉 ARM = Foundation,

👉 Bicep = Easier way to write ARM.



Summary Notes

- ARM Templates = **JSON-based IaC** for Azure.
- Sections: **\$schema, parameters, variables, resources, outputs**.
- Benefits: **Consistency, Automation, Repeatability, CI/CD**.
- Deployment methods: **CLI, PowerShell, Portal, Pipelines**.
- Advanced: **Linked Templates, Nested Templates, Functions**.

- **Bicep** = modern, simplified alternative (but uses ARM under the hood).