Week-11 | Azure Resource Manager (ARM) Templates in 60 Minutes | ARM Templates vs Bicep vs Terraform

Introduction to Azure Resource Manager (ARM)

What is Azure Resource Manager?

Azure Resource Manager (ARM) is the deployment and management service for Azure. It provides a consistent way to create, update, and manage resources in Azure. It enables users to:

- Deploy infrastructure as code (IaC).
- Apply access control using Role-Based Access Control (RBAC).
- Organize resources efficiently using **resource groups**.
- Enable **idempotent deployments** (same deployment yields the same result every time).
- Use **tags** to categorize resources for cost management and governance.

Why Use ARM Templates?

- **Infrastructure-as-Code (IaC)**: Allows declarative resource provisioning.

- **Consistency**: Ensures that deployments remain consistent across environments.
- **Automation**: Reduces manual work by automating deployments.
- **Repeatability**: Templates can be reused for multiple deployments.
- **Security & Compliance**: Integrates with Azure Policy for governance.

Understanding ARM Templates

What are ARM Templates?

ARM templates are JSON files that define the structure and configuration of Azure resources. They follow a **declarative** approach, meaning you specify **what** to deploy, and Azure takes care of **how** to deploy it.

Basic Structure of an ARM Template

An ARM template consists of several sections:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {},
  "variables": {},
  "resources": [],
  "outputs": {}
}
```


Key Components

1. **$schema**: Defines the template schema (versioned URL).
2. **contentVersion**: Specifies the version of the template (useful for versioning deployments).
3. **parameters**: Allows passing values dynamically at deployment time.
4. **variables**: Stores values that are reused multiple times in the template.
5. **resources**: Defines the actual Azure resources (VMs, storage accounts, databases, etc.).
6. **outputs**: Returns values after deployment (e.g., connection strings, resource IDs).

Working with ARM Templates

Creating an ARM Template for a Storage Account

```
{
```

```
  "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountName": {
      "type": "string",
      "minLength": 3,
      "maxLength": 24
    },
    "location": {
      "type": "string",
      "defaultValue": "eastus"
    },
    "sku": {
      "type": "string",
      "defaultValue": "Standard_LRS"
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2022-09-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('sku')]"
      },
      "kind": "StorageV2",
      "properties": {}
    }
  ]
}
```

Deploying ARM Templates

Deployment Using Azure CLI

1. **Create a resource group**
2. az group create --name MyResourceGroup --location eastus
3. **Deploy the ARM template**
4. az deployment group create --resource-group MyResourceGroup --template-file template.json
5. **Check deployment status**
6. az deployment group show --name MyDeployment --resource-group MyResourceGroup

Deployment Using PowerShell

1. **Login to Azure**
2. Connect-AzAccount
3. **Deploy the template**
4. New-AzResourceGroupDeployment -ResourceGroupName MyResourceGroup -TemplateFile template.json

---

ARM Templates vs. Bicep vs. Terraform

Introduction to Bicep

Bicep is a **domain-specific language (DSL)** for Azure that simplifies ARM templates. It provides an easier-to-read syntax while maintaining the same capabilities as ARM templates.

Example Bicep vs. ARM Template

Bicep Syntax (Storage Account Example)

```
param storageAccountName string

param location string = 'eastus'


resource storageAccount 'Microsoft.Storage/storageAccounts@2022-09-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
  kind: 'StorageV2'
```

```
}
```

Equivalent ARM JSON Template

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "storageAccountName": {
      "type": "string"
    },
    "location": {
      "type": "string",
      "defaultValue": "eastus"
    }
  },
  "resources": [
    {
      "type": "Microsoft.Storage/storageAccounts",
      "apiVersion": "2022-09-01",
      "name": "[parameters('storageAccountName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "Standard_LRS"
      },
      "kind": "StorageV2",
      "properties": {}
    }
  ]
}
```

Terraform for Azure

Terraform is an open-source Infrastructure-as-Code (IaC) tool that supports multiple cloud providers, including Azure. It offers better modularization, **state management**, and **multi-cloud** capabilities.

Example Terraform Configuration for an Azure Storage Account

```
provider "azurerm" {
  features {}
}

resource "azurerm_storage_account" "example" {
  name                     = "examplestoracc"
  resource_group_name      = "MyResourceGroup"
  location                 = "East US"
  account_tier             = "Standard"
  account_replication_type = "LRS"
}
```


Comparison Table: ARM vs. Bicep vs. Terraform

Feature	ARM Templates	Bicep	Terraform
Syntax	JSON (complex)	YAML-like (simplified)	HCL (Terraform DSL)
State Management	No state	No state	Uses Terraform state files
Reusability	Parameterized templates	Modular	Highly modular & reusable
Multi-Cloud Support	Azure only	Azure only	Multi-cloud (AWS, GCP, Azure)
Learning Curve	Steep	Easier than ARM	Easier with better documentation

Best Practices for ARM Templates

1. **Use Bicep Instead of JSON** – Simplifies writing and maintaining templates.
2. **Parameterize Templates** – Avoid hardcoding values; use parameters instead.
3. **Validate Templates Before Deployment** – Use az deployment group validate to catch errors early.
4. **Use Linked and Nested Templates** – Modularize large deployments for better manageability.
5. **Implement CI/CD Pipelines** – Automate deployments with GitHub Actions or Azure DevOps.
6. **Secure Sensitive Data** – Store secrets in Azure Key Vault instead of hardcoding them.
7. **Version Control** – Store templates in Git repositories like GitHub, GitLab, or Azure Repos.
8. **Use Tags for Cost Management** – Apply tags to resources to track and optimize cloud spending.

Conclusion

ARM templates provide a powerful and consistent way to deploy infrastructure in Azure. While JSON-based ARM templates are widely used, **Bicep simplifies** the deployment process with an improved syntax. Terraform, on the other hand, is **preferred for multi-cloud environments**.

For **Azure-only deployments**, Bicep is the best choice due to its **readability and maintainability**. For **multi-cloud strategies**, Terraform is **more flexible**.
