…

Contents

[Scenario Overview 2](#_Toc183531141)

[Technical Documentation of the Project 3](#_Toc183531142)

[Resources Used 3](#_Toc183531143)

[Implementation Steps 3](#_Toc183531144)

[Security Considerations 4](#_Toc183531145)

[Cost Optimization 4](#_Toc183531146)

[Monitoring and Maintenance 5](#_Toc183531147)

[Key Configurations 5](#_Toc183531148)

[Conclusion 5](#_Toc183531149)

[Deploying Azure Resources Using Bicep and Visual Studio Code (VS Code) 6](#_Toc183531150)

[Introduction to Bicep and Its Advantages 6](#_Toc183531151)

[Setting Up VS Code for Bicep Development 6](#_Toc183531152)

[Workflow for Deploying Resources with Bicep and VS Code 6](#_Toc183531153)

[Sample Bicep Code Used in the Project 7](#_Toc183531154)

[Advantages of Using Bicep in the Project 8](#_Toc183531155)

[Challenges and Solutions 8](#_Toc183531156)

[Conclusion 8](#_Toc183531157)

# Private and Local .NET Application with Automated Photo Upload Scenario

**Scenario Overview:** A company, *Global Media Corp*, is implementing a private and local .NET application hosted in Canada. This application will serve specific internal business needs. Additionally, a separate virtual machine located in Germany is set up to automatically upload approximately 1,000 photos daily to a solution in Canada. These photos are considered non-critical, so routine access to them is not necessary.

As an Azure Administrator, your task is to design and implement this scenario while ensuring that the .NET application can handle high-demand requests efficiently and that cost and security considerations are prioritized.

**Goals:**

1. **Deploy a Private .NET Application**: Implement a .NET application in Azure Canada that is accessible only from within the Germany region, ensuring that it can handle high-demand requests efficiently.
2. **Automate Photo Uploads**: Set up an automated solution for the virtual machine in Germany to upload photos daily to a specified location in Canada.
3. **Cost and Security Optimization**: Ensure that both the application and the photo upload process are cost-effective and secure, adhering to company policies regarding data security and expenditure.

**Student Notes:**

* **Local .NET Application Deployment**: Students should deploy the .NET application using Azure resources in the Canada region.
* **Automated Upload Process**: Design an automated process for the VM in Germany to upload photos to the Canadian storage solution.

**Important Note:**

* Please exclude **Azure WAF (Web Application Firewall)** and **Azure Firewall** from your security and architectural design for now. These can be postponed and should not be included in your initial implementation.

**Important Considerations:**

* **Cost Management**: Evaluate the costs associated with the chosen Azure services.
* **Security Practices**: Students should focus on securing the application and the data transfer process.

**Expectations:**

* **Application Deployment**: Students should successfully deploy the .NET application in the Azure Canada region.
* **Automated Upload Demonstration**: Students must show how the virtual machine in Germany automatically uploads photos to the Canadian solution.
* **Cost and Security**

# Technical Documentation of the Project

**Project Overview**

The goal of this project is to create a secure and automated system for uploading approximately 1,000 images daily from a Virtual Machine (VM) located in Germany to a Blob Storage Account in Canada. The project leverages Azure services, emphasizing cost efficiency, security, and automation. This documentation outlines the resources used, implementation steps, key configurations, and monitoring strategies.

## Resources Used

1. **Azure Virtual Machine (VM):**
   * **Purpose:** Acts as a Hybrid Worker for executing automation tasks.
   * **Region:** Germany.
2. **Blob Storage Account:**
   * **Purpose:** Stores the uploaded images.
   * **Region:** Canada Central.
   * **Tier:** Cool.
3. **Azure Automation Account:**
   * **Purpose:** Orchestrates the automated image upload using Runbooks.
4. **Azure Virtual Network (VNet):**
   * **Purpose:** Provides secure connectivity between resources.
   * **Regions:**
     + **Germany:** VNet with IP range 172.16.0.0/26.
     + **Canada Central:** VNet with IP range 10.0.0.0/26.
5. **Private Endpoint:**
   * **Purpose:** Ensures secure, private access to Web App.
6. **VNet Peering:**
   * **Purpose:** Connects VNets in Germany and Canada for data transfer.
7. **Web App:**
   * **Region:** Canada Central
   * **Purpose**: Simple .net for internal usage and access
8. **Network Security Groups (NSG):**
   * **Purpose:** Restricts traffic between VNets and resources.
   * Applied to VNets in both Germany and Canada with specific rules for image uploads.
9. **Log Analytics Workspace (Log W/S):**
   * **Purpose:** Monitors logs and metrics for all resources**.**
   * **Region:** Canada Central**.**

## Implementation Steps

1. **Create the Blob Storage Account:**
   * Use the Standard\_LRS replication type.
   * Enable a **Private Endpoint** to ensure secure access.
   * Configure a container named blob67 for storing images.
2. **Set up the Virtual Machine in Germany:**
   * Deploy a Windows 10 VM with managed identity enabled.
   * Install the required tools:
     + PowerShell 7.x
     + AzCopy for transferring images.
3. **Set up Azure Automation Account:**
   * Create an Automation Account in Germany.
   * Configure a **Hybrid Worker** on the VM to run PowerShell Runbooks.
4. **Establish VNet Peering:**
   * Peer the VNets in Germany and Canada.
   * Ensure bidirectional communication for secure data transfer.
5. **Develop PowerShell Runbook:**
   * Automate the image upload process using AzCopy.
   * Schedule the Runbook for daily execution.
6. **Apply Access Control with BLOB SAS token:**
7. **Configure Network Security Groups (NSG):**
   * In Canada, allow inbound traffic to the Blob Storage from Germany's VNet.
   * In Germany, restrict VM traffic to allow only outbound connections to the Canada VNet.
8. **Set up Log Analytics Workspace (Log W/S):**
   * Connect the VM and Automation Account to the Log Analytics Workspace.
   * Collect logs for Runbook executions and VM performance.

## Security Considerations

* **Private Connectivity:**  
  All communications occur over private links via VNets and Private Endpoints. Public access is disabled.
* **BLOB SAS token**
* **Encrypted Transfers:**  
  Data is transmitted securely using HTTPS.
* **Using NSG**

## Cost Optimization

1. **Virtual Machine:**
   * **Size:** Standard\_B1ms for minimal operational costs.
2. **Blob Storage Tier:**
   * Using the **Cool tier** to store non-critical images at a lower cost.
3. **Automation Account:**
   * Avoiding Logic Apps for cost efficiency in automation.
4. Log Analytics Workspace:
   * Optimize log retention policies to avoid excessive costs.

## Monitoring and Maintenance

1. **Log Analytics Workspace:**
   * Configured in Canada Central to centralize logs and metrics for all resources.
   * Tracks Runbook execution, VM performance, and storage metrics.
2. **Alerts:**
   * Configured alerts for failed uploads or anomalies in VM performance.

## Key Configurations

* **Blob Storage:**
  + Use of SAS Tokens for granular access control during uploads.
* **VNet Peering:**
  + Ensure proper routing configurations for seamless communication.
* **Runbook Execution:**
  + Triggered daily using a schedule in the Automation Account.
* **NSG Rules:**
  + Specific rules applied to allow and restrict traffic for the project’s requirements.

## Conclusion

This project successfully implemented a robust and scalable solution for automating image uploads in Azure. By integrating Azure services such as VNets, Private Endpoints, Automation Accounts, and Log Analytics Workspace, the solution ensures high security, efficiency, and cost-effectiveness. The system is ready to handle increased workloads with minimal adjustments.

# Deploying Azure Resources Using Bicep and Visual Studio Code (VS Code)

This section describes the step-by-step process for automating Azure resource deployments using Bicep and VS Code as implemented in the project. Bicep, a domain-specific language (DSL), simplifies the authoring and management of Azure Resource Manager (ARM) templates. Its seamless integration with VS Code makes it a powerful choice for infrastructure as code (IaC).

## **Introduction to Bicep and Its Advantages**

* **Bicep Overview:**  
  Bicep is a declarative language used for defining Azure resources. It provides simpler syntax compared to ARM templates while supporting all their features.
  + **Advantages of Bicep:**
    - Simplified and readable syntax.
    - Built-in support for resource dependencies.
    - Modular structure for reusable components.
    - Full integration with existing Azure CLI and PowerShell tools.

## **Setting Up VS Code for Bicep Development**

1. **Install VS Code:**  
   Download and install [VS Code](https://code.visualstudio.com/).
2. **Install the Bicep Extension:**
   * Open VS Code.
   * Go to the **Extensions** view (Ctrl+Shift+X).
   * Search for "Bicep" and install the official extension by Microsoft.
3. **Install Azure CLI:**
   * Download and install [Azure CLI](https://learn.microsoft.com/en-us/cli/azure/install-azure-cli).
   * Log in to your Azure account:
   * az login
4. **Bicep CLI Installation (Optional):**  
   If not bundled with Azure CLI, install Bicep manually:
5. az bicep install

## **Workflow for Deploying Resources with Bicep and VS Code**

1. **Write the Bicep Code:** Define resources in a .bicep file. Below is an example for creating a Storage Account:
2. resource storageAccount 'Microsoft.Storage/storageAccounts@2022-09-01' = {
3. name: 'mystorageaccount'
4. location: 'East US'
5. sku: {
6. name: 'Standard\_LRS'
7. }
8. kind: 'StorageV2'
9. }
10. **Validate the Bicep File:**  
    Use Azure CLI to ensure the code is error-free:
11. az bicep build --file main.bicep
12. **Deploy the Bicep Template:**  
    Deploy the resources to Azure:
13. az deployment group create --resource-group MyResourceGroup --template-file main.bicep

## **Sample Bicep Code Used in the Project**

1. **//Create a Virtual Network (VNet):**
2. resource vnet 'Microsoft.Network/virtualNetworks@2022-09-01' = {
3. name: 'vnet-canada'
4. location: 'Canada Central'
5. properties: {
6. addressSpace: {
7. addressPrefixes: [
8. '10.0.0.0/26'
9. ]
10. }
11. }
12. }
13. **//Create an App Service with a Private Endpoint:**
14. resource appService 'Microsoft.Web/sites@2022-03-01' = {
15. name: 'webApp-canada'
16. location: 'Canada Central'
17. properties: {
18. serverFarmId: appServicePlan.id
19. }
20. }
21. resource privateEndpoint 'Microsoft.Network/privateEndpoints@2022-09-01' = {
22. name: 'webAppPrivateEndpoint'
23. location: 'Canada Central'
24. properties: {
25. subnet: {
26. id: vnet.properties.subnets[0].id
27. }
28. privateLinkServiceConnections: [
29. {
30. name: 'appServiceConnection'
31. properties: {
32. privateLinkServiceId: appService.id
33. }
34. }
35. ]
36. }
37. }
38. **//Create a Virtual Machine (VM):**
39. resource vm 'Microsoft.Compute/virtualMachines@2022-08-01' = {
40. name: 'vm-germany'
41. location: 'Germany West Central'
42. properties: {
43. hardwareProfile: {
44. vmSize: 'Standard\_B1ms'
45. }
46. storageProfile: {
47. imageReference: {
48. publisher: 'MicrosoftWindowsDesktop'
49. offer: 'Windows-10'
50. sku: '20h2-ent'
51. version: 'latest'
52. }
53. osDisk: {
54. createOption: 'FromImage'
55. }
56. }
57. osProfile: {
58. computerName: 'vmGermany'
59. adminUsername: 'adminUser'
60. adminPassword: 'SecurePassword123!'
61. }
62. }
63. }

## **Advantages of Using Bicep in the Project**

1. **Readability and Maintenance:**  
   The simplified syntax allowed for better readability and easier collaboration among team members.
2. **Resource Dependencies:**  
   Automatic handling of dependencies reduced manual effort.
3. **Error Detection:**  
   VS Code’s IntelliSense and Bicep CLI validation helped identify issues during development.
4. **Reusability:**  
   Modular Bicep files enabled reuse of templates for similar projects.

## **Challenges and Solutions**

1. **Dependency Issues:**  
   Managing inter-resource dependencies was tricky initially. Using explicit dependency properties in Bicep resolved this.
2. **Deployment Errors:**  
   Errors like circular dependencies were addressed using modular file structures and clear resource names.

## **Conclusion**

Using Bicep with VS Code streamlined the process of resource deployment in this project. It enhanced productivity, reduced human errors, and provided a scalable framework for future deployments. The above workflow and examples can be used as a guide for similar Azure projects.