**ELC -Azure Policy Deployment using PowerShell and Azure DevOps Pipeline - Proof of Concept (POC)**

**Initial Draft**

**Submitted to**

Text, logo

Description automatically generated

**By**

A logo with colorful dots

Description automatically generated

**Wipro Technologies**

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Date of Revision** | **Description of Change** | **Reason for Change** | **Reviewed By** |
| 1.1 | 06-Nov-24 | Initial Draft | NA | Kannan & Felix |

**Author/Reviewer/Approvals**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Designation** | **Responsibility** | **Date** |
| Kannan Kuppusamy | Technical Lead | Reviewer |  |
| Felix Jebamani | Lead Consultant | Reviewer |  |
| Jamshid Abedi | ED, Global Head of Security Engineering & Operations | Approver |  |

**TABLE OF CONTENTS**

1. Objective
2. Scope
3. Azure Policy with Audit Mode
4. Azure Policy with Enforcement
5. Azure Policy with Exemption
6. Prerequisites
7. Solution Approach
   * Define Azure Policies
   * PowerShell Script for Deployment
   * Azure DevOps Pipeline Configuration
8. Validation
9. Expected Outcome
10. Conclusion
11. Future Enhancements

**1.Objective:**

The objective of this POC is to demonstrate an automated approach to deploying multiple Azure policies using PowerShell scripts within an Azure DevOps pipeline. This ensures consistent policy enforcement across multiple subscriptions or resource groups, reducing manual efforts and minimizing configuration drift. Additionally, it aims to provide a framework for enterprises to streamline policy governance, improve compliance, and integrate DevSecOps best practices.

**2. Scope This POC covers:**

* Writing PowerShell scripts to deploy Azure policies in bulk.
* Automating policy deployment using Azure DevOps pipelines.
* Validating deployed policies in Azure.
* Providing a scalable and repeatable solution.
* Addressing policy assignment across multiple scopes (Management Groups, Subscriptions, Resource Groups).
* Ensuring role-based access control (RBAC) compliance within Azure environments.
* Enhancing security posture by enforcing enterprise-wide policies.

### **3. Azure Policy with Audit Mode**

* The audit effect is used to create a warning event in the activity log when evaluating a non-compliant resource, but it doesn't stop the request.
* Azure Policy JSON Script with Audit Mode: -

**6. Prerequisites**

**To implement this POC, the following prerequisites must be met:**

* Azure Subscription: Ensure that an active Azure subscription is available with Owner or Policy Contributor permissions.
* Azure DevOps Account: Set up an Azure DevOps organization with a connected repository (Azure Repos, GitHub, etc.).
* Service Principal: A service principal with Microsoft.Authorization/policyAssignments/write permissions to deploy and assign policies.
* PowerShell Modules: Install and configure Az.PolicyInsights and Az.Resources PowerShell modules.
* YAML Pipeline Configuration: Define a pipeline structure within Azure DevOps to automate policy deployment.
* Policy Definition Repository: Store predefined policy JSON files in the version control repository.
* Network Access: Ensure firewall rules allow communication between Azure DevOps and Azure services.

**7. Solution Approach**

* Consider Deployable Resources: Before enforce any policy need to specify the scope of the policy, like here you need to specify the resources that you can deploy using the Azure policy. You canspecify the set of Virtual Machines, Storage Accounts, Key Vaults, etc that under your organization can deploy.
* Consider Location Restrictions: You can also specify the locations where the resources can be deployed. You can specify the locations like the regions or geographical locations which are available to your organizations.
* Consider Rules Enforcement: You can specify the rules and the configuration options under which the resources can be deployed in your organization. You can also specify the required tags for the resources that are deployed in your organization.
* Consider Inventory Audits: You can use the Azure policy here with Azure Backup Service over your workloads and run inventory audits to make sure that the resources are compliant with the policies that you have enforced in your organization while ensuring the backup of your resources.

There are four basic steps which are required to create the Azure policies:

* *Create the policy definition:* A policy definition is just like a logic which needs to be satisfied by the resources to perform certain actions in your organization. You can either create your own policy definitions or you can use the built-in-policy definitions. For example, you can create a policy definition to specify the locations where the resources can be deployed in your organization.
* *Create an initiative definition*: An initiative definition is a collection of multiple policy definitions which help you to keep log of the compliance of your resources and also gives you a wider scope to manage the policies. You can either create your own initiative definitions or you can use the built-in initiative definitions. You can use the initiative definition to verify and ensure your resources are compliant with the security requirements.
* *Scope of the initiative definition*: Azure Policies lets you to control how the initiative definitions are enforced to your resources in your organization. In the same you can also specify the scope of the initiative definition to the specific management groups, subscriptions or resource groups.
* *Determine Compliance*: As you allot the initiative definition, where you can check the compliance of the resources in your organization. Individual resources, resource groups and subscriptions within a scope can be unchecked from the compliance of the initiative definition. Exclusions are handled individually for each initiative definition.

**Step 1: Define Azure Policies**

Policies are defined in JSON format and stored in a repository (e.g., GitHub or Azure Repos). These policies define compliance requirements, such as blocking public IP addresses, enforcing encryption, and ensuring resource tagging.

Example Policy - Deny Public IP:

{

"properties": {

"displayName": "Deny Public IP",

"policyType": "Custom",

"mode": "All",

"policyRule": {

"if": {

"field": "Microsoft.Network/publicIPAddresses",

"exists": "true"

},

"then": {

"effect": "Deny"

}

}

}

}

**Step 2: PowerShell Script for Deployment**

A PowerShell script is created to deploy policies in bulk. This script iterates over JSON files and creates policies in Azure.

$policies = Get-ChildItem -Path "./Policies" -Filter "\*.json"

foreach ($policy in $policies) {

$policyContent = Get-Content -Path $policy.FullName -Raw

New-AzPolicyDefinition -Name $policy.BaseName -Policy $policyContent -Mode All

}

**Step 3: Azure DevOps Pipeline Configuration**

A YAML-based pipeline is created to automate policy deployment and integrate it into CI/CD workflows.

trigger:

- main

pool:

vmImage: 'windows-latest'

steps:

- task: AzurePowerShell@5

inputs:

azureSubscription: 'Service Connection'

ScriptType: 'FilePath'

ScriptPath: './Scripts/DeployPolicies.ps1'

azurePowerShellVersion: 'Latest'

**8. Validation**

* Verify the policies in the Azure portal under Policy Definitions.
* Check policy assignments and ensure they are applied correctly.
* Validate policy compliance state using Azure Policy Compliance Dashboard.
* Review pipeline execution logs for errors or warnings.
* Use PowerShell or Azure CLI commands to validate policies:
* Get-AzPolicyDefinition | Where-Object { $\_.Properties.DisplayName -eq "Deny Public IP" }
* Confirm that non-compliant resources trigger enforcement actions.

**9. Expected Outcome**

* Automated bulk deployment of Azure policies across multiple environments.
* Improved governance and compliance enforcement.
* A repeatable and scalable deployment process for policy management.
* Reduction in manual configuration errors.
* Centralized policy management and visibility through Azure Policy.
* Seamless integration into DevOps workflows to ensure security compliance.

**10. Conclusion**

This POC demonstrates the feasibility of automating Azure policy deployment using PowerShell and Azure DevOps pipelines. By integrating this approach into enterprise workflows, organizations can ensure consistent policy enforcement across their Azure environments, enhancing security, compliance, and operational efficiency. Additionally, this approach reduces human error, improves deployment efficiency, and provides better visibility into governance compliance.

**11. Future Enhancements**

* Automated Policy Assignments: Implement role-based assignment automation to ensure policies are applied at scale.
* Integration with Monitoring Tools: Use Azure Monitor, Log Analytics, and Security Center to track compliance status.
* Multi-Environment Support: Expand support for multiple environments (Development, Test, Production) with environment-specific policies.
* Infrastructure as Code (IaC) Enhancements: Extend policy management using Terraform or Bicep templates.
* Automated Rollback Mechanism: Implement rollback functionality to revert to previous policy configurations in case of deployment failure.
* Policy Version Control: Enable tracking and rollback of policy changes using GitOps methodologies.
* Enhanced Reporting: Generate compliance reports and integrate with dashboards for real-time insights.