ChatGPT

# Plan for Implementing NIST SP 800-53 Rev. 5 Controls with Azure Policy (EPAC)

## Overview and Objectives

This plan outlines how to map **NIST SP 800-53 Revision 5 technical controls** to Azure Policy definitions and implement them via **Enterprise Policy as Code (EPAC)**. The goal is to move from a **monitoring-only** posture (policies in Audit mode, disabled) to **active enforcement** of all applicable controls. This involves enabling Azure built-in policy definitions (or creating custom ones where gaps exist), setting appropriate **policy effects** (Deny, Modify, Append, DeployIfNotExists, etc.), and ensuring remediation tasks can run with a managed identity. The outcome will be an EPAC configuration that **enforces NIST 800-53 controls** wherever technically possible, using Azure Policy as the mechanism.

## Current State and Gaps

- **Current EPAC Codebase:** Deploys the built-in **NIST SP 800-53 Rev. 5** policy initiative (PolicySet) at the desired scope (e.g. management group or subscription). However, policies are currently set to *AuditIfNotExists* or *Audit*, and many are effectively **disabled** (not enforcing changes). For example, controls like limiting subscription owners or requiring managed identities are only audited and not prevented ① ②.
- **Gap: No enforcement** – non-compliant resources are allowed since all policies are auditing or disabled. Also, **remediation** (DeployIfNotExists/Modify) policies are not executing because a **managed identity** is not configured for the assignment. This leaves many technical controls unaddressed beyond reporting.

## Mapping NIST 800-53 Controls to Azure Policies

First, identify which NIST 800-53 Rev.5 controls can be automated with Azure Policy. Microsoft provides a mapping of NIST controls to Azure Policy definitions ③ ④. Not every control has a direct one-to-one policy, but all **technical controls** (those dealing with system configurations, access, monitoring, etc.) are candidates for policy enforcement. Key categories include:

- **Access Control (AC):** e.g. Account management, access enforcement, least privilege. **Azure Policy solutions:** Restrict number of subscription owners (AC-2) ①, require Azure AD admins for SQL servers (AC-2) ⑤, enforce use of Managed Identity for apps (AC-3) ⑥, disable local authentication keys for Cognitive Services (AC-3) ②, enforce RBAC on Kubernetes (AC-3(7)), etc. Many are covered by built-in policies (some default to Audit). We will enable these and set to **Deny** or **Deploy** as appropriate.
- **Audit & Accountability (AU):** e.g. logging of activities, audit storage. **Azure Policy solutions:** Ensure **diagnostic logs** and **resource logs** are enabled for all services (AU-2/AU-12) – e.g. Key Vault, SQL, Storage account logs should be turned on ⑦ ⑧. Use *DeployIfNotExists* policies to configure

diagnostic settings to send logs to Log Analytics (or Audit if auto-deployment is not feasible without specifying a workspace). Also enforce log retention settings where applicable.

- **Configuration Management (CM):** e.g. secure configurations, baseline configurations. **Azure Policy solutions:** Enforce use of approved VM images or configurations (deny unauthorized OS versions or SKUs), require **Guest Configuration** extension on VMs to check OS settings [9], ensure VMs have a system-assigned identity (so Guest Configuration can run) [10], enforce tagging standards (Append tags for classification if required). Azure *Guest Configuration* policies can audit in-VM settings like password policy or encryption protocols, which map to CM/SI controls. These will be deployed and enforced via DeployIfNotExists (for the extension) and Audit/Deny for non-compliance findings.
- **Contingency Planning (CP):** e.g. data backup (CP-9). **Azure Policy solutions:** Ensure all critical resources have backups enabled. The built-in NIST initiative includes an audit policy for VM backup [11], but we will move to **enforcement**. Azure Backup has built-in **DeployIfNotExists** policies to auto-enable backup on VMs (configuring them to a Recovery Services vault) [12]. We will incorporate these to automatically back up VMs (or at minimum, flag non-backed VMs and optionally deny VM creation if backup isn't enabled, depending on strategy). Database services' geo-backup settings (for SQL, MySQL, etc.) are audited by built-ins [13] [14]; those that can only be set at creation will be enforced by *Deny* at deploy time if possible (e.g. deny creating a DB instance without geo-redundant backup if policy allows).
- **Identification & Authentication (IA):** e.g. strong authentication, key management. **Azure Policy solutions:** Enforce Azure AD integration where possible (e.g. require Azure AD login for VMs instead of local accounts, use AAD for AKS, etc.), ensure MFA (this is mainly handled via Azure AD Conditional Access – outside Azure Policy's scope, so we note it as a gap not solved by Azure Policy). Enforce **credential management** practices: e.g. all Key Vault keys/secrets must have an expiration date (built-in policy for KV, mapping to IA-5) – set this to Deny new secrets without expiry. Enforce disabling default or insecure accounts (built-in guest config policy can audit Linux accounts without passwords [15]).
- **System & Communications Protection (SC):** e.g. network security, encryption in transit and at rest. **Azure Policy solutions:** Implement policies to **disable public network access** on resources (storage accounts, databases, cognitive services, etc.), requiring Private Link or service endpoints (maps to SC-7 boundary protection). Many built-ins exist (e.g. "Public network access should be disabled for [Service]" – we'll set these to **Deny** so that internet-exposed resources cannot be created). Enforce **encryption in transit** – require HTTPS/TLS1.2 for web apps, functions, storage, etc. [16] [17] (set these to Deny so non-TLS or legacy TLS are not allowed). Enforce **encryption at rest** – by default Azure encrypts data at rest, but for certain services require customer-managed keys (if mandated by compliance). Built-ins for CMK usage exist (e.g. Cosmos DB, Azure Storage, Azure SQL etc. "should use customer-managed keys") [18] [19]. We will decide case-by-case if requiring a CMK (set Audit or Deny per organizational policy) – for high sensitivity, set to Deny creation of resources without CMK. Otherwise, at least Audit so it's tracked [20]. Also enforce secure network configuration: ensure NSGs on subnets/VMs, no broad open ports (use custom Azure Policies to *deny* NSG rules that allow 0.0.0.0/0 on admin ports, mapping to AC-17 remote access requirements).
- **System & Information Integrity (SI):** e.g. malware defense, monitoring. **Azure Policy solutions: Enable Defender for Cloud** on all resources – policies audit if services like Defender for VMs, SQL, Storage are enabled [21] [22]. We will enforce turning these on where possible. (Azure Policy can't directly turn on a Defender plan via deploy effect yet, but we can treat it as *must be enabled* – possibly integrate with Azure Security Center's auto-provision or use a DeployIfNotExists if available). Ensure **anti-malware extension** on Windows VMs (built-in policy exists to audit existence of Endpoint Protection agent). We will deploy the Microsoft Antimalware extension to all Windows VMs using DeployIfNotExists. Likewise, ensure Linux VMs have Endpoint protection or are covered by

Defender for Linux. Additionally, enable **Azure Monitor agents** on VMs (via Guest Configuration policy that audits if Log Analytics agent or Azure Monitor agent is installed [23]). These address SI-4 (monitoring for indications of compromise) and SI-3 (malicious code protection).

*Note:* Some NIST controls are procedural or not fully enforceable via technical means. Those will remain partially addressed (marked as **Manual, Disabled** in Azure's mapping [4]). Our focus is on **all controls that can be enforced or audited by Azure Policy**. Azure Policy compliance will give a partial view of NIST compliance [4], so any control that requires documentation or processes beyond Azure's scope should be handled separately (outside this plan).

## Enforcement Strategy with Policy Effects

To move from audit-only to enforcement, we will leverage the full range of Azure Policy *effect* capabilities. Each policy or initiative parameter will be configured to the strongest applicable effect:

- **Deny:** Use Deny to **block non-compliant resources** from being created or updated. This is suitable for controls where any violation must be prevented. For example, deny creation of storage accounts with public access or without encryption, deny enabling insecure protocols (SSH password auth, HTTP, older TLS) [24] [16], deny use of disallowed VM images or SKUs, etc. Many built-in policies include a parameterized effect that supports "Deny" (instead of just Audit) [2]. We will set those to Deny in the EPAC assignment parameters for strong enforcement.
- **Modify:** Use Modify to **automatically configure resources at creation** time to be compliant. Example: a policy to *add a system-assigned managed identity* to any new VM (if it lacks one) uses the Modify effect [10]. This satisfies prerequisites for Guest Configuration (mapping to AC-3). We will enable such policies so that new VMs get a managed identity transparently. Another example is enabling required encryption settings on a resource by modification if supported. Modify effects will be turned on for policies like "Add secure transfer to storage accounts" where Azure Policy can flip a setting on the resource during deployment.
- **Append:** Use Append to **add required settings** to a resource if not specified by the user. This is useful for adding mandatory tags (e.g., an "Owner" or "Classification" tag for CM-8 inventory or RA-2 data classification) or adding properties like required IP filtering rules. We will review if any NIST control requires adding metadata or config that can be appended. For instance, if organizational policy mandates a specific tag for all resources, an Append policy will ensure that tag is added if missing. (There may not be many built-in NIST policies using Append by default, so this may involve creating a custom policy if needed for tagging requirements.)
- **DeployIfNotExists (DINE):** Use DeployIfNotExists to **remediate missing configurations by deploying required resources/settings**. This is critical for controls that require the presence of some service or configuration. For example: deploy the **Guest Configuration extension** to all VMs (so in-guest policies can work) [25], deploy required **diagnostic settings** (log configuration) for services that don't have them, or enable backup on existing VMs by deploying a backup configuration. We will enable built-in DINE policies such as: "Deploy Guest Configuration extension on Linux/Windows VMs" [25] [26], "Configure diagnostic settings to Log Analytics workspace" (requires specifying a target Log Analytics workspace), and use Azure Backup's built-in DINE policies for auto-enabling VM backups. These policies will be set with effect *DeployIfNotExists* (and necessary parameters like target Log Analytics workspace ID, Recovery Services vault name, etc.). After assignment, Azure will attempt to auto-deploy the needed resource if it's missing, and mark the resource compliant once deployed.

- **Audit (and AuditIfNotExists):** For controls that **cannot be automatically fixed or enforced without risk**, we will retain Audit but ensure they are active (not disabled). Audit will log the compliance state for review and further manual action. For example, some policies might remain Audit if denial is not feasible (e.g. “maximum 3 owners per subscription” – Azure Policy can detect >3 owners [1], but it cannot automatically remove them, and denying role assignments isn’t natively supported by Azure Policy). Such policies will be left as Audit, but they will be *enabled* so that compliance data is collected and alerts can be raised. We will document any remaining manual control requirements that Audit covers (so the organization can address them procedurally).

**Effect Parameterization:** The NIST 800-53 Rev.5 initiative has many policies with an `effect` *parameter* that can be set per assignment [2]. Using EPAC, we will override these parameters to the desired effect values (e.g., from “Disabled” or “Audit” to “Deny” or “DeployIfNotExists”). Not every policy supports every effect – we will use the allowed values. For instance, if a policy only supports Audit or Disabled, we might need to replace it with a similar custom policy that can Deny, or use a different approach. In the assignment configuration, we will set parameters like `"mysqlDisablePublicNetworkAccessEffect": "Deny"` or `"kvRsaCryptographyMinimumKeySizeEffect": "Deny"` as needed [27]. The EPAC tool allows specifying these in a parameters JSON or via a CSV for bulk changes.

**Testing enforcement:** We must be cautious that enabling Deny/Modify effects can potentially block legitimate deployments if misconfigured. We will test each critical deny policy in a lower environment to ensure it only blocks what it should. For example, after switching a policy to Deny, attempt to create a resource violating that policy – it should be rejected (as the Medium example showed for Azure SQL public access being denied after effect change [28] [29]). If a policy doesn’t support Deny (or it’s not effective), we might need to create a custom policy definition to enforce that control.

## Implementation Plan (Step-by-Step)

Below are the concrete steps for the AI coder to implement these changes in the EPAC codebase:

**Step 1: Update EPAC Assignment to Include NIST 800-53 Rev.5 Initiative**
If not already done, ensure the EPAC configuration is assigning the built-in **NIST SP 800-53 Rev.5** policy set at the appropriate scope (e.g. management group). In EPAC’s JSON, this can be done via a `definitionEntry` or `definitionEntryList` pointing to the built-in initiative ID. The initiative’s ID is `179d1daa-458f-4e47-8086-2a68d0d6c38f` [30]. For example, in EPAC you might have:

```
"definitionEntryList": [
  {
    "policySetId": "/providers/Microsoft.Authorization/policySetDefinitions/
179d1daa-458f-4e47-8086-2a68d0d6c38f",
    "displayName": "NIST SP 800-53 Rev. 5",
    "assignment": {
      "name": "nist-800-53-r5",
      "displayName": "NIST SP 800-53 Rev. 5 Initiative",
      "description": "Enforce NIST SP 800-53 Rev.5 technical controls via Azure
Policy."
    }
  }
]
```

This will deploy the entire set of NIST-linked policies. (If the codebase already includes this, ensure it's using the latest built-in version so we have all relevant definitions).

**Step 2: Configure Managed Identity for Policy Remediation**

Policies with **DeployIfNotExists** or **Modify** effects require an **assigned Managed Identity** to execute the remediation tasks [31] [32]. In EPAC, we must configure the assignment to use a managed identity and set up role assignments. Two approaches:

- **System-assigned identity:** Easiest option. In EPAC's settings, define `managedIdentityLocations` for the assignment (e.g., choose a region like `"*": "eastus2"` for all environments) [33]. EPAC will then create a system-assigned identity for the policy assignment in that region. EPAC automatically computes which built-in roles are needed by reading the policy definitions' `roleDefinitionIds` (each DeployIfNotExists has a list of required role IDs) [32]. EPAC will create the necessary Role Assignments for the identity with reason "Role Assignment required by Policy" [34]. This means the identity will get rights such as **Monitoring Contributor** for log deployment, **Backup Contributor** for backup deployment, **Virtual Machine Contributor** for VM extensions, etc., as needed by each policy. (Behind the scenes, for example, a policy to deploy diagnostic settings has a roleDefinitionId for Monitoring Contributor [35], so EPAC will assign that to the identity at the appropriate scope).
- **User-assigned identity (optional):** If the organization prefers a user-assigned managed identity, EPAC supports that [36]. You would create a user-assigned identity manually, assign it the required roles, and reference it in EPAC's config under `userAssignedIdentity`. Unless there is a specific need, we will use system-assigned for simplicity and let EPAC handle the roles.

Make sure to include in the EPAC assignment JSON something like:

```
"managedIdentityLocations": {
  "*": "eastus2"
}
```

This ensures the identity is provisioned. After deployment, verify that the identity has been created and the EPAC tool output shows role assignments being made (for example, Monitoring Contributor, Backup Contributor, etc.). Without this, any DeployIfNotExists policies will only flag non-compliance but cannot auto-remediate. With the identity, those policies will **automatically deploy fixes** when triggered (or via a remediation task).

**Step 3: Determine Effect Overrides for Each Policy**

Next, decide on the enforcement level (effect) for each control's policy. We will use a **CSV-driven approach** for manageability, as recommended by EPAC for large initiatives [37]. The process:

1. **Generate Parameter CSV:** Use EPAC to export the current parameters of the NIST 800-53 assignment to a CSV. This CSV will list each policy (often by its parameter name or policy name) and the current effect/value.

2. **Review and Set Effects:** For each policy line in the CSV, set the `Effect` column to the desired value for each environment. For example:
3. Set policies controlling **network exposure** to `Deny` (e.g., public access policies for Storage, SQL, Cosmos DB, etc. – change from Audit to Deny).
4. Set **encryption requirement** policies to `Deny` if non-compliance is unacceptable (e.g., require TLS 1.2: Deny if not using latest TLS; require disk encryption: Deny if someone tries to disable encryption).
5. Set **backup policies** to use `DeployIfNotExists` (for those that support it) or potentially `Deny` on new VM creation if backup cannot be auto-enabled. We will likely use Azure Backup's built-in deploy policy outside the initiative, rather than modifying the audit policy, since the audit policy doesn't have a deploy mode [12]. (In EPAC we can include the backup deploy policy separately with its parameters).
6. Set **Defender/Monitoring policies** to `DeployIfNotExists` where available (e.g. some policies might support deploying the agent or enabling the plan). If no deploy option, keep as Audit or consider `Deny` for new resource creation if missing security (for instance, optionally deny creation of a VM if it's not going to be covered by Defender – though enabling it is better than denying VM).
7. **Guest Configuration**: Ensure the policies "Deploy Guest Configuration extension on VMs" are enabled. These might already have effect fixed as DeployIfNotExists in the initiative (check if they were disabled by parameter; if so, set them active). Also ensure "Add system-assigned identity to VMs" (Modify effect) is enabled. These are crucial for enforcing in-guest settings for many controls (like password policies, etc.).
8. **Manual controls**: For any policy that is marked "Manual, Disabled" (purely documentation, e.g. "develop procedures"), leave them disabled (they can't be automated).

9. **Parameters for specific controls**: Some policies have additional parameters (like allowed locations, tag names, key size minimums, retention days, etc.). Use the CSV or JSON parameters to set those as well. For example, if NIST requires certain retention (AU-11), set "logRetentionDays" parameter to the required value in the assignment. Or if IA-5 requires passwords to be 14 characters, ensure any Guest Config policy checking password complexity has that configured if applicable.

10. **Apply the CSV in EPAC:** In the EPAC assignment JSON, reference the CSV for parameters and effect overrides (the EPAC docs allow linking a CSV so that EPAC will import those as parameters) [37]. EPAC will then build the assignment with all those effect settings. Alternatively, one can manually fill the `parameters` section in JSON for each policy's effect, but that is tedious. The CSV approach is preferred for clarity.

For illustration, an excerpt of the parameters section might look like:

```
"parameters": {
   "storageAccountPublicAccessEffect": "Deny",
   "sqlServerPublicNetworkAccessEffect": "Deny",
   "appServiceHttpsOnlyEffect": "Deny",
   "appServiceLatestTlsVersionEffect": "Audit",
   "azureKeyVaultAllowTrustedAzureServicesEffect": "Deny",
   "enforceBackupOnVMsEffect": "Disabled",
    "...": "..."
  }
```

*(Note: The actual parameter names will match the built-in definitions; the CSV will provide the exact keys. EPAC will ignore any parameters not applicable to a policy set [38] .)*

After configuring, the assignment will effectively turn on enforcement. For example, setting `mysqlDisablePublicNetworkAccessEffect = "Deny"` means any attempt to create an Azure Database for MySQL with public access enabled will be blocked by policy [27]. Setting `AzureContainerRegistryPublicNetworkAccessEffect = "Deny"` will block ACR without firewall. All such changes directly map to NIST requirements for restricting unauthorized network access (SC controls).

**Step 4: Add Custom Policies for Uncovered Controls (if needed)**
Review if any NIST technical controls are not addressed by the built-in policies. If gaps exist, create **custom Azure Policy** definitions to fill them. Potential custom policies and assignments could include:
- **Restricting Legacy Protocols or Instances**: e.g. disable creation of Azure Classic resources (if any still allowed) to meet modernization requirements. Built-ins already suggest migrating to ARM for VMs/Storage [39] which we'll enforce (Deny classic resources).
- **Least Privilege Enforcement**: Azure Policy cannot fully enforce RBAC best practices, but we might create an *Audit* policy to flag any Owner role assignments outside a central group, or flag overly permissive custom roles (some built-in audit exists for custom roles usage [40] ). We will ensure those are active (Audit).
- **Azure AD Policies**: Some IA controls (like MFA, password policies) are enforced in Azure AD/Entra ID, not via Azure Policy. We will note these as out-of-scope for EPAC and ensure they are handled via Azure AD Conditional Access or Identity Governance, rather than trying to use Azure Policy.
- **Alerting**: While not exactly a "deny", for some Audit policies we might want immediate alerting. We can attach Azure Monitor alerts on policy non-compliance results for critical controls (this is outside EPAC, but part of operationalizing the enforcement).

If custom policies are created, add them to the EPAC codebase (likely as JSON in the custom definitions folder) and include them in the assignment (via `definitionEntry` entries). For each custom policy, also specify an effect (preferably Deny or DeployIfNotExists) and include any `roleDefinitionIds` if using deploy, so EPAC can assign roles. The aim is that **all controls that can be automated are either covered by a built-in or a new custom policy** in the assignment.

**Step 5: Deploy and Test in Non-Production**
Before rolling out broadly, deploy the updated EPAC policy assignment to a test subscription or staging environment:
- **Validation of Assignment:** Confirm in Azure Portal that the NIST 800-53 Rev.5 initiative is showing as assigned and the parameters reflect the new effect values (you can view the assignment -> Parameters to see overrides). The UI should list many policies with effect = Deny now, instead of Audit [28] .
- **Managed Identity check:** In Azure Policy -> Assignments, the NIST assignment should show a managed identity principal ID. Check that this identity has the expected role assignments (Portal: Azure AD -> Roles and Administrators -> find the identity, or via PowerShell/CLI). You should see roles like *Monitoring Contributor*, *Backup Contributor*, *Virtual Machine Contributor*, *Network Contributor* (if needed for NSGs), etc., assigned at relevant scopes. If any DeployIfNotExists policy is failing to remediate and logging permission errors, that means a missing role – add it either via EPAC additionalRoleAssignments or manually and then update EPAC config. (EPAC usually covers built-in roleDefinitionIds automatically [32] , but if a policy deploys

across resource group boundaries or to another subscription, you may need an additional assignment as noted in EPAC docs [41] .)
- **Resource Deployment Tests:** Attempt to create resources that violate key policies to ensure they are blocked. For example, try to create a storage account with public blob access enabled – it should fail with a policy denial stating the policy name. Try creating a SQL server without an Azure AD admin – if we set that to Deny, the deployment should be blocked (or if we used DeployIfNotExists, the policy might automatically add an AAD Admin if configured to do so). Verify VMs created get a managed identity automatically (they should, due to the Modify effect). Create a VM and see if the Guest Config extension gets auto-installed (it might take a few minutes after creation; or trigger a remediation). Also test that a non-compliant configuration triggers a compliance result: e.g., disable soft-delete on a Key Vault if policy requires it – the vault should show as non-compliant and ideally the policy could auto-enable it if we set a DeployIfNotExists for that.
- **Backup auto-enable:** Create a new VM in test scope; verify that after assignment, either the VM is automatically being backed up (if we used the backup deploy policy) or at least flagged. If using deploy, an initial policy remediation run might be needed to kick off backup enabling. Ensure the Recovery Services vault has the VM registered.
- **Remediation of existing resources:** For resources that existed prior to assignment, run **on-demand remediation tasks** for DeployIfNotExists policies (either via Azure Portal's "Remediate" button or Azure CLI). This will apply the changes (e.g., install extensions on existing VMs, enable backup on existing VMs, etc.). With EPAC, you can also re-run the pipeline to create role assignments first, then manually trigger remediation if needed. All existing non-compliances should either auto-remediate or at least show up so admins can address them.

Record any issues and adjust the policy parameters or definitions as needed. For example, if a Deny is too broad (blocking something unintended), consider switching that particular policy to Audit or refining its conditions.

### Step 6: Production Rollout and Ongoing Management

After testing, promote the changes to the production EPAC configuration:
- Deploy the assignment at the production scope. Immediately, it will start enforcing new deployments. Monitor for any deployment errors or incidents (in case something critical was blocked unexpectedly – be prepared with a break-glass procedure to exclude a scope or change a parameter to Audit if a critical production need arises).
- Monitor the **Azure Policy compliance dashboard** for NIST SP 800-53. Over time, the compliance % should increase as non-compliant resources are remediated or denied. Use continuous monitoring to ensure all policies remain effective. For any **"Audit" residual policies**, create internal tickets or processes to address those findings (since Azure Policy can't auto-fix them).
- Keep the mapping document updated: maintain a spreadsheet or document that lists each NIST control, how it's implemented (Policy name/effect or "manual"), and the compliance status. This helps demonstrate coverage to auditors.
- **Tune as necessary:** If Azure introduces new built-in policies for NIST or updates (they may update mappings over time [42] ), incorporate those. Also, if new Azure services are in use, ensure they are covered by some policy (for instance, if adopting a new PaaS service, look for policies to enforce its security config).
- **Environment-specific adjustments:** You might choose to have slightly different settings in non-prod (e.g., Audit in dev, Deny in prod for some policies). EPAC's parameter CSV can handle multiple columns for different environment types (dev/test/prod). Use that to your advantage. For example, in a dev subscription

you may set a policy to Audit so developers can test something, whereas prod is strict Deny. This should be documented in the CSV and EPAC config (using its environment selector logic).

By following these steps, **all NIST SP 800-53 technical controls that Azure can enforce will be actively enforced or auto-remediated**, moving beyond just audit. Azure Policy (with EPAC) will help ensure the cloud environment stays in compliance with the NIST baseline. Controls that are not automatable will at least be monitored via audit policies (or handled through organizational processes). This plan provides the AI coder with a clear blueprint to implement the necessary changes in the EPAC codebase and achieve end-to-end compliance mapping.

## References and Supporting Information

- Microsoft's mapping of NIST 800-53 Rev.5 controls to Azure Policy definitions [4] [2]
- Azure built-in **NIST SP 800-53 Rev.5** initiative definition ID and usage [30] [28]
- Azure Policy effect types and requirement for managed identities (for DeployIfNotExists/Modify) [31] [32]
- EPAC documentation on managed identities and using CSV for bulk parameter management [33] [43]
- Azure Backup policy for VMs – audit vs. enforce via DeployIfNotExists [12]

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [39] [40] [42]
Regulatory Compliance details for NIST SP 800-53 Rev. 5 - Azure Policy | Microsoft Learn
https://learn.microsoft.com/en-us/azure/governance/policy/samples/nist-sp-800-53-r5

[12] Audit and enforce backup during VM creation automatically using Azure Policy - Azure Backup | Microsoft Learn
https://learn.microsoft.com/en-us/azure/backup/backup-azure-auto-enable-backup

[27] [32] [33] [34] [36] [37] [38] [41] [43] Policy Assignment Files - Enterprise Policy As Code (EPAC)
https://azure.github.io/enterprise-azure-policy-as-code/policy-assignments/

[28] [29] [30] Assessing and Enforcing NIST SP 800–53 Compliance with Azure Policy and Terraform | by Robbie Douglas | Medium
https://medium.com/@robbiedouglas/assessing-and-enforcing-nist-sp-800-53-compliance-with-azure-policy-and-terraform-77ba60c3b884

[31] Azure Policy definitions deployIfNotExists effect - Azure Policy | Microsoft Learn
https://learn.microsoft.com/en-us/azure/governance/policy/concepts/effect-deploy-if-not-exists

[35] Azure Policy DeployIfNotExists is not adding diagnostic setting ...
https://learn.microsoft.com/en-au/answers/questions/1533730/azure-policy-deployifnotexists-is-not-adding-diagn