AVATIER™

White Paper

Avatier Least-Privilege Permissions for Hybrid Identity Service Accounts (Zero Trust Compliance)

Published
March 20, 2025

Avatier Least-Privilege Permissions for Hybrid Identity Service Accounts (Zero Trust Compliance)

Introduction: In a hybrid Active Directory (AD) environment with Azure AD (Microsoft Entra ID), an identity management **service account** often acts as a bridge for user and group management. To align with **Zero Trust** principles, this account must have only the minimal permissions required to perform its tasks – no more, no less. This ensures compliance with frameworks like NIST SP 800-53, ISO 27001, HIPAA, GDPR, SOC-2, and SOX, all of which mandate strict access control and **least privilege** practices (least privilege - Glossary | CSRC) (7 Security Controls You Need For General Data Protection Regulation (GDPR) - ProcessUnity). The service account here will provision/deprovision users, manage groups, reset passwords (including self-service resets), and update attributes/roles via Microsoft Graph API, without being a Domain Admin or Global Admin. The following report details the minimal AD permissions needed, best practices for securing the account, threats of over-privilege and mitigations, and how to integrate with Microsoft Graph API while preserving least privilege.

1. Minimal Active Directory Permissions (Least Privilege Access)

The table below summarizes the **required on-premises Active Directory permissions** for the service account. Each permission is scoped as narrowly as possible (e.g. to specific Organizational Units or object classes) to enforce least privilege. Justifications and compliance relevance are included:

AD Permission	Justification	Compliance Relevance
Create User Accounts (delegated on specific OU(s)) – *e.g. allow "Create" user objects in a given OU*	Needed for **user provisioning** – the account can create new user objects in designated OU(s) (but not elsewhere). This avoids using Domain Admin rights for user creation.(Permissions required for the AD account configured in ADManager Plus) The account should also have Read access on relevant attributes to populate new accounts.	Limits scope of account creation to intended areas, supporting the principle of least privilege (NIST SP 800-53 AC-6) ([least privilege - Glossary
Delete/Disable User Accounts (delete user objects in specific OU; write `userAccountControl` to disable)	Allows **deprovisioning** by deleting or disabling accounts. Delegating only on the OU with standard users prevents modifying accounts outside scope (e.g. administrators). Disabling is done by writing the *account control flag*, without broader rights.	Ensures terminated users' access is promptly removed in a controlled way, aligning with SOX and ISO 27001 requirements to revoke access when no longer needed (ISO 27001 - Annex A.9 - Access control – ISO Templates and Documents Download). Granular delegation avoids excessive rights, satisfying GDPR's data minimization by limiting who can alter personal data.
Modify User Attributes (write specific attribute sets on user objects)	Required for updating user profiles (e.g. titles, department, phone) and role-related flags. The service account gets Write permission on needed user attributes (or "Write All Properties" on users in OU if fine-grained control isn't feasible ([Microsoft Entra Connect: Accounts and permissions – Microsoft Entra ID	Microsoft Entra Connect: Accounts and permissions This does not include critical admin attributes (like adding to Domain Admins group).
Reset User Passwords (and unlock accounts)	Allows the service to perform **password resets** for users (e.g. during self-service password reset or helpdesk-assisted reset). Delegated "Reset Password" on user objects (and related permissions to clear lockout) is granted ([Microsoft Entra Connect: Accounts and permissions - Microsoft Entra ID	Microsoft Entra Connect: Accounts and permissions No other admin rights are included, and ideally not granted on admin accounts if not needed.
Group Management (Create, Delete Groups in specific OU or scope)	Needed for provisioning/deprovisioning of groups (e.g. creating new role groups or distribution lists). The account is delegated rights to create and delete group objects in a specific OU (or specific group scope). It does not have broad group privileges beyond that (avoids touching built-in or privileged groups). Permissions required for the AD account configured in ADManager Plus	Enforces *separation of duties* by scoping group creation to the IAM service's domain. Complies with **ISO 27001 Annex A.9** by restricting group management to authorized processes. Auditors (SOX, SOC-2) will see that only this service (and no extra admins) can create groups, and only in defined areas, reducing risk..
Manage Group Membership (modify 'member' property on groups)	Allows adding or removing users from groups for **role assignments and access management**. Delegate **Write** on the 'member' attribute for specific security and Microsoft 365 groups that the service manages. For example, the account might have permission to update membership for all non-privileged groups (but not add users to sensitive groups like Domain Admins).	This granular right ensures the service can assign roles (often represented by group membership) without full group admin rights. Aligns with **least privilege** by limiting the account to just membership changes it needs. GDPR and other privacy laws benefit since only authorized role changes occur, and any improper addition to a group (which might grant access to data) is less likely to happen without broad privileges.

Read Directory Data (users/groups)	(*Usually granted by default Domain Users membership*.) The service account needs read access to relevant AD objects to query existing users, groups, and attributes (for detecting changes, verifying account status, etc.). By default a normal user can read most AD data; ensure this is intact or explicitly grant read rights on OUs if inheritance was broken.	Read access is fundamental and poses low risk, but it's noted for completeness. It adheres to **least privilege** since only reading does not modify data. Many compliance frameworks (NIST, ISO) don't restrict read access heavily, but they do require monitoring – so all read operations by the service account should still be auditable (supporting SOC-2 security and GDPR transparency/accountability).
No Elevated Admin Privileges (Explicitly **not** a Domain Admin or Enterprise Admin)	Rationale: The account should not be a member of built-in high-privilege groups like Domain Admins, Account Operators (if possible), Schema or Enterprise Admins. All required functions can be accomplished with specific delegated rights as above. This prevents the service account from becoming an all-powerful backdoor.	Removing blanket admin rights is critical for **Zero Trust**. It ensures **segregation of duties** and satisfies regulatory demands to limit privileged access. NIST and ISO frameworks explicitly advise against using accounts with broad privileges when not needed. Many breaches (e.g. OPM 2015, Marriott 2018) were worsened because attackers exploited over-privileged service accounts ([Three Cyberattacks Where Compromised Service Accounts Played a Key Role.

Table Note: Each permission above should be granted via AD's **delegation** model on the narrowest scope (e.g. a specific OU for user accounts and groups dedicated to the identity management process). This adheres to the Zero Trust idea of *"never trust, always verify"* – the service account should be treated like any user and only trusted with the bare minimum it needs. In practice, Microsoft's own tooling (e.g. Entra Connect) follows this model by assigning a custom account only the needed read/write/reset permissions instead of domain admin (Microsoft Entra Connect: Accounts and permissions - Microsoft Entra ID | Microsoft Learn). The result is compliance with multiple frameworks' requirements for least privilege: NIST calls for "minimum necessary" access (least privilege - Glossary | CSRC), ISO 27001 requires that privileged rights be tightly controlled and reviewed (ISO 27001 - Annex A.9 - Access control – ISO Templates and Documents Download), and regulations like **HIPAA** and **GDPR** mandate limiting access to sensitive data to only those with a justified need (7 Security Controls You Need For General Data Protection Regulation (GDPR) - ProcessUnity).

2. Security Best Practices for the Service Account

Even with minimal permissions, how the service account is **configured and managed** is crucial. Following security best practices will harden the account against misuse or compromise. Below are key best practices (drawn from Microsoft recommendations and real-world lessons):

- **Use Dedicated, Non-Interactive Account:** Treat the service account as strictly a machine/service identity – it should *not* be used by any human for interactive login (no GUI/RDP access). Deny interactive logon rights via Group Policy. This limits exposure of credentials (attackers can't phish something that isn't used to log in) and aligns with the principle that service accounts are for services only.
- **Apply the Principle of Least Privilege:** Ensure the account has **only** the permissions listed in the table above – nothing more. Do not make it a Domain Admin or give it blanket rights. As one case study noted, over-provisioning (like making a service account a domain admin) can be "very dangerous" and, if compromised, lets an attacker "own the domain". Always prefer granular AD delegation and minimal Azure roles to fulfill the necessary tasks.
- **Strong Authentication and Credential Management:** Use a long, complex password (or certificate/secret) for the account, managed in a secure vault. If possible, leverage **Managed Service Accounts (MSA/gMSA)** for on-prem AD services – these have automatic password rotation and cannot be used to interactively logon, boosting security. In Azure AD, consider using a **managed identity or service principal** with a certificate instead of a raw password. This reduces the risk of weak or leaked passwords. (Managed identities also benefit from Azure's control plane protections).
- **Restrict Logon Scope:** Limit where the service account can log on or be used. For example, if the account is only needed by a particular sync service server or API proxy host, restrict its logon to that host (using the "Log On To" account restriction in AD for user accounts, or by designing it as a gMSA tied to specific servers). This way, if an attacker somehow obtains the credentials, they cannot use them on arbitrary systems.
- **Enable Monitoring and Auditing:** Continuously **monitor the service account's activity**. Enable AD auditing for actions like account creation, group changes, and password resets initiated by this account. Unusual usage (e.g. the service account trying to log on interactively or modify an out-of-scope object) should trigger alerts. Real-world breaches have gone undetected for months because service account activity wasn't closely watched

Three Cyberattacks Where Compromised Service Accounts Played a Key Role | Silverfort). Using security information and event management (SIEM) tools or Azure AD reporting, set up alerts for anomalies – e.g. if the account attempts to access domain controllers or if there are multiple failed password attempts (could indicate misuse).

Use Separation of Duties (SoD): Where feasible, **separate the roles** even for the service operations. For instance, use one service account (or app registration) for user provisioning/deprovisioning and a different one for password reset/self-service functions. This way, each account has an even smaller set of permissions, and a compromise in one area (say the password reset service) cannot directly impact user provisioning. This SoD approach is recommended in Zero Trust to minimize blast radius. It also helps with compliance audits, since each account's purpose and permissions are clearly defined and can be reviewed independently.

Follow Microsoft's Tiering Model: If applicable, place the service account in the appropriate **AD security tier** (Tier 1 or 2, not Tier 0). Microsoft's Active Directory tier model suggests keeping accounts that administer endpoints or manage users separate from those that administer the directory itself. The service account here administers regular users and groups, so it should not have rights over Tier 0 assets (domain controllers, etc.) and ideally should be treated as a Tier 1 asset. This prevents credential reuse or attack paths between tiers (an attacker compromising this account shouldn't be able to directly jump to controlling AD infrastructure).

Implement Multi-Factor for Console Access: While service accounts by design aren't used interactively, if the identity management solution provides a console or portal where this account's credentials are needed (for example, to configure the Graph API connector), protect that console with multi-factor authentication. Also, if using an Azure AD service principal, protect it with Conditional Access (e.g. only allow token requests from specific IPs or require certificate-based auth). This ensures that even if the credential is stolen, an attacker cannot easily abuse it from an unknown location.

Regular Reviews and Rotation: Periodically review the service account's permissions and group memberships (at least quarterly or per compliance requirement) (ISO 27001 - Annex A.9 - Access control – ISO Templates and Documents Download). Remove any access that is no longer necessary (principle of **access recertification**). Also rotate credentials regularly – even gMSA passwords rotate automatically, but for an Azure AD app client secret or certificate, ensure it's renewed before expiration and old credentials are invalidated. Regular review meets ISO 27001's control A.9.2.5 (user access rights review) and NIST recommendations for continuous monitoring.

Document and Train: Clearly document what this service account is for, what permissions it has, and who is responsible for managing it. In many organizations, service accounts get created and years later no one remembers their scope – which is dangerous. Keep an updated document (or use an identity governance tool) to track this account. Train the administrators of the identity management system on these best practices so they don't unintentionally expand the account's privileges during troubleshooting. Real-world case studies have shown that "forgotten" service accounts with lingering high privileges are common weaknesses (Three Cyberattacks Where Compromised Service Accounts Played a Key Role | Silverfort).

By implementing these best practices, the service account remains tightly controlled. Microsoft's guidance echoes these points: **"grant privileges sparingly"** (least privilege), prefer managed identities for services , and monitor continuously since service accounts are a common target. Many high-profile breaches (like SolarWinds) have prompted companies to reinforce such measures on service accounts to meet the stringent requirements of frameworks like SOC-2 and to bolster their **Zero Trust** posture.

3. Potential Threats and Mitigations

A misconfigured or over-privileged service account can become a significant threat vector. Below, we outline key threats related to identity service accounts and how to mitigate them, ensuring compliance and minimizing risk:

- **Threat: Compromise of Service Account Credentials.** If an attacker obtains the service account's password or token, they could **abuse its permissions**. Over-privileged accounts make this worse – for example, a service account that is a domain admin would let an attacker fully control AD. Case studies underscore this risk: in the **Marriott breach**, attackers obtained credentials of privileged service accounts with domain admin access and used pass-the-hash techniques to move laterally and access sensitive systems (Three Cyberattacks Where Compromised Service Accounts Played a Key Role | Silverfort). In the **OPM breach** (2015), a stolen contractor login led to misuse of a service account that had high privileges, enabling theft of millions of personnel records (Three Cyberattacks Where Compromised Service Accounts Played a Key Role | Silverfort). **Mitigations:** As discussed, *limit the account's privileges* so even if compromised, damage is contained. Implement strong authentication (complex passwords, possibly tied to hardware security modules or managed identities to prevent credential dumping). Monitor login locations – if the account suddenly authenticates from an unusual host or IP, lock it down. Utilize Azure AD Conditional Access or on-premises firewall rules to allow the service account's activities only from expected systems. Regular password rotations and preventing the account from being used for interactive logons also reduce the window of opportunity for attackers.
- **Threat: Lateral Movement and Privilege Escalation.** An attacker who compromises the service account might use it as a stepping stone to escalate privileges in the hybrid environment. For instance, with improper rights, they might modify other accounts or change group memberships to grant themselves admin access. In a hybrid scenario, if the service account also has significant Azure AD privileges, the attacker could possibly create backdoors in cloud apps or read sensitive cloud data. **Mitigations:** Use *role separation* – the service account should not have any admin rights beyond its narrow purpose (no membership in administrators groups). Ensure it cannot modify administrative groups or accounts. Use tools like Microsoft's Privileged Access Management (PAM/PIM) to put just-in-time controls around any privilege that is highly sensitive (e.g., if the service account ever needs to perform an elevated task, require a time-bound approval). In Azure AD, avoid giving the account permanent Global Administrator; if absolutely needed for role assignments, consider using Azure AD Privileged Identity Management (PIM) so that role assignment privileges are enabled only on approval and for a limited time. This way, even if an attacker gets in, they cannot easily escalate without tripping alarms or needing additional approvals.
- **Threat: Abuse of Graph API Permissions or Tokens.** If the service account is an Azure AD application (service principal) with Graph API permissions, a malicious actor could exploit those **API permissions** if they gain the app's secret or certificate. Unlike user accounts, app permissions (like `User.ReadWrite.All`) are not limited by Conditional Access and can be very powerful across the tenant. For example, an attacker could call Graph API to create accounts or exfiltrate user data. **Mitigations:** Treat application secrets like production passwords – store them in secure vaults, rotate regularly, and prefer certificates over client secrets for better management. Use Azure AD's **App Consent** policies to ensure only pre-authorized apps exist. Also, utilize Azure AD's capability to **disable or reconfigure credentials** quickly if compromise is suspected (e.g., delete the app's secret to invalidate tokens). Monitor Azure AD sign-in logs and audit logs for the service principal's usage; Azure can alert on anomalous app activities. When possible, use **managed identities** (which have no static secret at all) for cloud-to-cloud integration to eliminate the secret theft risk. Finally, scope Graph permissions to least privilege (detailed in the next section) – e.g. if the app doesn't need to read all mailbox data, don't grant it; this limits what an attacker can do with a stolen token.
- **Threat: Lack of Visibility and Shadow Admins.** Service accounts are often not subject to the same scrutiny as interactive admins. An organization might forget that this account exists or not realize it has accumulated permissions over time (the so-called "shadow admin" issue where an account effectively has admin abilities without being in obvious groups). Attackers count on these accounts being **unmonitored** (Three Cyberattacks Where Compromised Service Accounts Played a Key Role | Silverfort). If logging and alerting are weak, the attacker can use the service account for a long time without notice (as happened in some breaches). **Mitigations:** Perform regular audits of all accounts with elevated permissions, including service accounts. Use tools or scripts to enumerate what

each service account can do (review AD ACLs and Azure role assignments for that account). Implement a strict joiner-mover-leaver process for service accounts akin to employees – e.g., if the service's scope changes, update or remove its rights. Leverage conditional access and risk-based sign-in detection in Azure AD – even though service principals don't log in like users, Azure AD Identity Protection can detect certain anomalies. Also, ensure that the knowledge of the service account isn't confined to one person; have multiple people aware and a process to update its credentials in a controlled way (to avoid situations where emergency changes lead to over-provisioning).

- **Threat: Non-Compliance or Audit Failure.** From a governance perspective, having an account with excessive rights can lead to compliance violations. Auditors checking for SOX or **SOC-2** compliance, for example, will flag accounts that have broad access to financial systems or sensitive personal data without proper justification. They may also flag lack of periodic access reviews. **Mitigations:** Document the account's purpose and permissions for auditors. Demonstrate that it has the minimum required access. Map each permission to a control objective (as we did in the table) to show alignment with regulatory requirements. Implement quarterly access reviews (and retain evidence of these reviews). Use logs to prove that the account's activities are appropriate (e.g., show that all its group changes correspond to approved ticket requests from HR provisioning, etc.). By maintaining this level of governance, you both reduce the risk of actual misuse and satisfy compliance mandates.

In summary, the **risks** of a service account revolve around it becoming a single point of failure if misused. The **mitigations** center on least privilege, monitoring, and strict governance. The Zero Trust mindset of "assume breach" is useful – assume this account *could* be compromised, and architect its access such that even in that event, the blast radius is limited and alarms go off. Microsoft's real-world incident analyses (like the SolarWinds post-mortem) have led to advisories that underscore many of these mitigations, from tightening service account ACLs to deploying comprehensive monitoring (Three Cyberattacks Where Compromised Service Accounts Played a Key Role | Silverfort). Implementing these will help prevent the service account from becoming an attack pathway and ensure continued compliance with security frameworks.

4. Integration with Microsoft Graph API (Hybrid Identity Permissions)

In a hybrid environment, the identity service account often operates both on-premises (AD) **and** in Azure AD via the Microsoft Graph API. This dual nature requires careful permission design in both contexts:

- **On-Premises AD Permissions:** As outlined in Section 1, the account uses a delegated AD user (or gMSA) with rights to create/modify users, manage groups, and reset passwords in specific OUs. This account (let's call it **"AD provisioning account"**) is typically used by the sync/provisioning service running on-prem. For example, an identity management tool might run as a service on a Windows server using this account's context to call LDAP/AD. The key is that this account's ACL in AD is limited to the scope of identity data. Microsoft's Azure AD Connect, for instance, uses an AD Connector account with *"read/write all properties"* on users/groups and password reset rights, but nothing more (Microsoft Entra Connect: Accounts and permissions - Microsoft Entra ID | Microsoft Learn). This ensures it can sync and writeback identities without being a domain admin. The same model should be followed for any custom hybrid connector.
- **Azure AD (Graph API) Permissions:** The service will act as a **Graph API client** to perform cloud-side operations (provisioning/deprovisioning cloud identities, group management in Azure AD, etc.). Here we have two choices:
    1. **Use an Azure AD Application (App Registration) with Application Permissions:** This means creating an app registration for the identity management service and granting it Graph API **application permissions** like *User.ReadWrite.All* and *Group.ReadWrite.All*. These permissions allow the app to **create, update, and delete** any user or group in the tenant (since application permissions run as a trusted service, not as a specific user). If role assignments (Azure AD directory roles) need to be managed, the app might also require

RoleManagement.ReadWrite.Directory (which allows reading and assigning directory roles). **Important:** Only grant the scopes absolutely required – for example, if the service never manages Azure AD admin role assignments, do **not** consent to the RoleManagement permission. Microsoft's guidance is clear: *"request the least privileged permissions that your app needs... requesting more than necessary is poor security practice."* (Microsoft Graph permissions reference - Microsoft Graph | Microsoft Learn). So an app that just needs to sync user profiles should not also get permissions to read all mail or Intune devices, etc. Admin consent is required for these high-level scopes, so during that process one should double-check the scopes.

Once granted, these app permissions are tenant-wide. Mitigation: use Azure AD's Administrative Units if possible to limit scope – however, note that app permissions in Graph currently ignore Administrative Units (they're all-tenant). If that level of granularity is needed, consider using delegated permissions with a service account user instead.

2. **Use a Dedicated Service Account in Azure AD with Delegated Permissions:** In this model, instead of an app with its own identity, you have an actual Azure AD user (or synced AD account) that is assigned a specific **Azure AD role** (or roles). The Graph API calls are then made **in the context of that user** (delegated flow, e.g. using OAuth with username/password or certificate via OAuth ROPC or a similar technique). The advantage here is you can leverage Azure AD's built-in roles to enforce least privilege. For instance, to manage users and groups, you could assign the account the **"User Administrator"** role in Azure AD. A User Administrator can create and manage users (and groups) but cannot touch roles like Global Admin (Least privileged roles by task - Microsoft Entra ID | Microsoft Learn) (Least privileged roles by task - Microsoft Entra ID | Microsoft Learn). According to Microsoft's role documentation, User Administrators can also reset passwords for non-admin users and manage group settings (Least privileged roles by task - Microsoft Entra ID | Microsoft Learn) (Least privileged roles by task - Microsoft Entra ID | Microsoft Learn), covering our use case. If you need the service to also manage certain admin role assignments, consider the **"Privileged Role Administrator"** role, which allows managing role assignments without being Global Admin. Keep in mind that granting that is powerful, so only do so if absolutely necessary, and possibly use a separate account for that function.

   When using delegated permissions via Graph, the effective permissions are the **intersection** of the Azure AD role and the Graph OAuth scope. For example, if the account is granted the User Administrator role, and your Graph OAuth consent is for `Directory.AccessAsUser.All` (which allows the app to do anything the signed-in user can do in the directory), then the account's role limits the actual operations. In our scenario, that works well: the OAuth scope is broad but the role is narrow, so the account can only perform what a user admin can. Microsoft's Graph provides some specific delegated permissions for certain tasks – e.g., to call the **password reset API** (`authenticationMethods/resetPassword`), the delegated permission `UserAuthenticationMethod.ReadWrite.All` is needed *and* the account must have an appropriate role like Authentication Administrator or Password Administrator (authenticationMethod: resetPassword - Microsoft Graph v1.0 | Microsoft Learn). Alternatively, a User Administrator can reset passwords through another Graph endpoint by updating the passwordProfile (though that requires the `Directory.AccessAsUser.All` scope in Graph) (How to Reset Or Update User Passwords with Microsoft Graph In ...). The key is aligning the Graph permission with the role:

   - For **user create/update**: `User.ReadWrite.All` (delegated or application) or use `Directory.AccessAsUser.All` (delegated) with User Admin role. (Least privileged roles by task - Microsoft Entra ID | Microsoft Learn)
   - For **group management**: `Group.ReadWrite.All` or rely on User Admin role which includes group management (Least privileged roles by task - Microsoft Entra ID | Microsoft Learn).

-
    - For **password reset**: no application permission exists (Graph does not allow apps without a user context to reset passwords for security reasons), so the service must use delegated and the account must have e.g. Authentication Administrator or be a User Admin resetting a non-admin user (authenticationMethod: resetPassword - Microsoft Graph v1.0 | Microsoft Learn).
    - For **reading directory data**: `Directory.Read.All` (or included in the above).
    - For **role assignments** (if automated): `RoleManagement.ReadWrite.Directory` as an app permission, or the service account user having Privileged Role Administrator role (and using delegated `Directory.AccessAsUser.All`). This is the *highest risk* permission and should be avoided unless absolutely mandated by business requirements.
- **Hybrid Flow Considerations:** In many hybrid setups, Azure AD Connect or a cloud provisioning service will handle creating the AD account and synchronizing to Azure AD. If your identity management service creates users in Azure AD *and* you also have AD on-prem, you need to decide the source of authority. Commonly, new users are created in on-prem AD (via the delegated rights) and then sync to Azure AD. Alternatively, some organizations use cloud-first provisioning (create in Azure AD via Graph) and then use **AD Writeback** (a feature of Entra ID) to create the AD account. If doing the latter, ensure the Azure app has permission to call the writeback (which is usually through the Microsoft Graph API to create an on-premises synced user – this requires the Entra hybrid synchronization service to allow inbound writes). Azure AD Connect's cloud provisioning agent, for example, uses a specific **"Directory Synchronization Accounts"** role in Azure AD that grants the necessary Graph permissions to write to AD (Detecting and mitigating Active Directory compromises | Cyber.gov.au) – but that is a special case not broadly available for custom apps. Most likely, you'll provision in AD and let it flow to AAD.
- **Conditional Access for Hybrid Service:** If the service account is an Azure AD user, secure it with **Conditional Access**. For instance, require that sign-ins by this account occur only from a specific IP range (your data center or cloud VPC) and/or require device compliance if it ever interactively logs in (which should not happen often). If it's an app registration, use Azure AD's **named locations** or service tags to restrict where the OAuth token can be requested from (this can be done by associating the app to a conditional access policy using workload identities protection preview features). This adds an extra layer of security on the Graph side.
- **Logging and Graph Auditing:** Azure AD provides an audit log for directory changes. Ensure that all changes coming from this service principal or account are being logged (they should, with entries like "User created by X" where X is the service principal). Regularly review these logs or set up an automated job to reconcile them with expected changes (for example, if the service is supposed to only update users during business hours, an audit log entry of it doing something at 3 AM on a Saturday is suspect). Additionally, Graph API offers the ability to **capture audit events** for provisioning (if using SCIM or Graph provisioning APIs). Use these to demonstrate compliance – you can show auditors a trail of every group change and who/what made it.

To illustrate minimal Graph permissions, consider an example: If using an Azure AD app, you might end up granting **only** two application permissions – **User.ReadWrite.All** and **Group.ReadWrite.All** – to cover all needed user/group operations. This would require admin consent, but it is far more constrained than giving the app *Directory.ReadWrite.All* (which includes many other object types) or, worse, making it a Global Admin. Microsoft explicitly notes to avoid over-scoping because admins may hesitate to consent overly broad apps (Microsoft Graph permissions reference - Microsoft Graph | Microsoft Learn). By tailoring the Graph scopes and pairing them with a limited-role account, you achieve a **least-privilege hybrid integration**.

Conclusion: Integrating with Graph API in a Zero Trust fashion means *mirroring the least-privilege approach on-prem in the cloud*. Use the smallest set of Graph permissions or Azure roles that allow the service to function, and no more. Protect those credentials with the same rigor as on-prem credentials. By doing so, the service account will meet compliance mandates across the board – from NIST's emphasis on minimizing attack surface to GDPR's demands of controlled access – while still efficiently managing identities across the hybrid environment.

Sources: The recommendations and requirements above are supported by Microsoft's documentation and industry case studies. For instance, Microsoft's **"Implementing Least-Privilege Administrative Models"** and **Graph API permission guides** stress minimizing privileges (Microsoft Graph permissions reference - Microsoft Graph | Microsoft Learn), and real breach analyses highlight the dangers of excessive service account rights (Three Cyberattacks Where Compromised Service Accounts Played a Key Role | Silverfort). By adhering to these principles, organizations can confidently use a hybrid identity management service account that is both effective and compliant with frameworks like ISO 27001, SOC-2, HIPAA, GDPR, NIST, and SOX.