Agent Identity and Blueprint

Dennis Seah (dennis.seah@microsoft.com)

Microsoft Corporation

February 26, 2026

Version 0.9.0

Abstract

This document presents an approach to provisioning and governing agent identities in Azure Entra ID. It addresses the end-to-end workflow, beginning with the creation of agent blueprints that define metadata such as display names, descriptions, and tags, followed by the registration of corresponding application objects and service principals via the Microsoft Graph API. It then covers provisioning user-assigned managed identities in Azure and establishing federated identity credentials that enable passwordless authentication through OpenID Connect. Additional sections describe how to configure identifier URIs and expose custom OAuth2 permission scopes, as well as how to assign Azure RBAC roles—such as Storage Blob Data Reader—to grant agents least-privilege access to cloud resources. Reference PowerShell scripts accompany each topic, offering ready-to-use automation for every stage of the process.

Revision History

Version	*Date*	*Description*
0.9.0	2026-02-26	Initial draft

Contents

1 Introduction

This document provides guidance on agent identity and blueprints, including the motivation for agent identity, the concept of a blueprint, and how they relate to each other.

To run the scripts provided in this document, you need the following prerequisites:

- PowerShell and the Microsoft Graph PowerShell SDK. Follow the instructions in the official documentation to set up your environment.
- An Azure tenant with the appropriate permissions to create and manage agent identities and blueprints.
- The Microsoft Graph PowerShell SDK installed and configured on your local machine.
- The necessary permissions to authenticate and interact with the Microsoft Graph API, such as `AgentIdentityBlueprintPrincipal.ReadWrite.All` and `AgentIdentityBlueprint.AddRemoveCreds.All`.

The scripts provided in this document are intended for reference purposes. They can be adapted and integrated into your existing workflows and automation pipelines as needed. Always follow security best practices when managing identities and credentials in Azure Entra ID, including the principle of least privilege and secure storage of sensitive information.

Run the scripts in this order (change directory to `scripts/` first):

- Create an agent blueprint using `CreateAgentBlueprint.ps1`. Set the `$tenant_id` and `$blueprint_name` parameters (e.g., `$tenant_id = "00000000-xxxx-0000-xxxx-000000000000"` and `$blueprint_name = "my-blueprint"`). Sample output:

  ```
  blueprint_name    : my-blueprint
  blueprint_app_id  : <redacted>
  blueprint_obj_id  : <redacted>
  principal_id      : <redacted>
  secret_value      : <redacted>
  secret_id         : <redacted>
  secret_expiry     : 2/26/2027 1:48:06 PM
  user_id           : <redacted>
  ```

  Use `DeleteAgentBlueprints.ps1` to delete the blueprint.

- Create an agent identity using `CreateAgentIdentity.ps1`. Set `$agent_id_name`, `$blueprint_app_id`, `$client_secret`, and `$sponsor_user_id` (e.g., `$agent_id_name = "my-agent-identity"`). The remaining values can be obtained from the output of the previous script. Sample output:

  ```
  agent_id_name      : my-agent-identity
  agent_id_app_id    : <redacted>
  agent_id_id        : <redacted>
  blueprint_app_id   : <redacted>
  tenant_id          : <redacted>
  ```

  Use `DeleteAgentIdentities.ps1` to delete the agent identity.

- Create a user-assigned managed identity using `CreateUserManagedIdentity.ps1`. Set `$subscription_id`, `$mi_name`, `$rg_name`, and `$location` (e.g., `$mi_name = "my-managed-identity"`). The `$location` parameter defaults to `eastus2`. Sample output:

  ```
  ManagedIdentityName : my-managed-identity
  principal_id        : <redacted>
  client_id           : <redacted>
  resource_id         : <redacted>
  rg_name             : <redacted>
  location            : eastus2
  ```

  Use `DeleteUserManagedIdentity.ps1` to delete the user-assigned managed identity.

- Create a federated identity credential using `CreateFederatedIdentityCredential.ps1`. Set `$subject` (the `PrincipalId` of the managed identity created in the previous step), `$fed_cred_name` (a name of your choice for the federated identity credential), `$issuer` (the Azure AD endpoint for your tenant, e.g., `https://login.microsoftonline.com/<tenant_id>/v2.0`), and `$blueprint_app_id` (from the first step). Sample output:

```
fed_cred_id           : <redacted>
fed_cred_name         : my_fed_cred
blueprint_app_id      : <redacted>
subject               : <principal_id of the managed identity>
issuer                : https://login.microsoftonline.com/<tenant_id>/v2.0
audiences             : {api://AzureADTokenExchange}
```

  Use `DeleteFederatedIdentityCredential.ps1` to delete the federated identity credential.

- Add an identifier URI and scopes to the blueprint using `ConfigureIdentifierURIandScopes.ps1`. Set `$identifier_uri` (e.g., `api://my-blueprint`), `$scope_name` (e.g., `read.all`), `$scope_desc` (a description of the scope), `$scope_display_name` (the display name), `$scope_type` (`User` or `Admin`, defaults to `User`), and `$blueprint_app_id` (from the first step). Sample output:

```
blueprint_app_id : <redacted>
identifier_uri   : api://<blueprint_app_id>
scope_name       : read.all
scope_id         : <redacted>
scope_type       : User
tenant_id        : <redacted>
```

  Use `RemoveIdentifierURI.ps1` and `RemoveScope.ps1` to remove the identifier URI and scopes, respectively.

- To verify the setup, assign a role to the agent identity and access an Azure resource using the federated identity credential. For example, grant the agent identity the *Storage Blob Data Reader* role on a storage account using `SetupStoragePermissions.ps1`, then use the federated identity credential to obtain an access token and read blob data. Use `UnassignStoragePermission.ps1` to remove the role assignment.

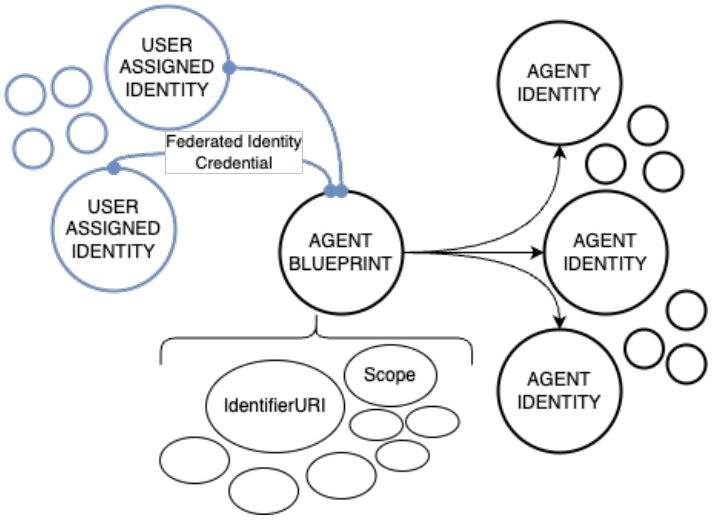

Figure 1: Composition of an agent blueprint

Figure 1 illustrates the composition of an agent blueprint. A blueprint functions as a declarative template that defines the configuration and permissions governing agent identities. At its core, the blueprint comprises an application registration in Azure Entra ID and may encompass multiple agent identities provisioned from

it. Each agent identity is bound to its parent blueprint and inherits the associated configuration, including permissions and federated identity credentials. This hierarchical model enables centralized, consistent management of agent identities and their access to Azure resources.

2 Azure Agent Blueprint

In *Azure Entra ID*, an *agent blueprint* is a reusable template that defines the configuration, capabilities, and permissions for a specific type of autonomous agent. Every agent identity in *Azure Entra ID* must originate from a blueprint, which acts as the parent management object. The blueprint specifies the App Roles and Microsoft Graph permissions (scopes) that all instances created from it are allowed to request, preventing individual agents from drifting into over-privileged states. Crucially, blueprints hold the credentials used for authentication. Individual agent instances do not carry their own secrets; they use the blueprint to request access tokens, which reduces *service principal sprawl*. If an administrator disables a blueprint, it immediately prevents the creation of new agents and can be used to halt functionality for all associated child identities across the organization. Once a blueprint is deleted, all child identities are also deleted, and any tokens issued to those identities become invalid.

Currently, creating and managing agent blueprints in *Azure Entra ID* requires familiarity with *PowerShell* and the *Microsoft Graph PowerShell SDK*. However, we envision a future state where agent blueprints can be programmatically created, managed, and rotated at scale, with seamless integration into the agent development lifecycle and DevOps pipelines. This would empower organizations to rapidly onboard new agents, enforce security best practices automatically, and maintain robust governance as the number of agents grows.

2.1 Sample PowerShell Script for Creating an Agent Blueprint

`scripts/CreateAgentBlueprint.ps1` is a sample script that demonstrates how to create an agent blueprint in *Azure Entra ID* using PowerShell. The script requires the following parameters: `$tenant_id` (the Azure tenant ID where the blueprint will be created) and `$blueprint_name` (the display name for the new blueprint). These can be passed as script parameters or set as session variables. The script authenticates to Microsoft Graph with the necessary permissions, checks for existing blueprints with the same name, constructs the request body with the specified blueprint name and current user as sponsor and owner, and issues a POST request to the Microsoft Graph beta endpoint to create the blueprint. It serves as a reference implementation for provisioning agent blueprints, which can be further automated in the future.

Upon successful creation, the script outputs the app ID and object ID of the new blueprint. A service principal is automatically provisioned for the blueprint, and its object ID is included in the console output. Note that blueprint credentials are not retrievable after creation; ensure that the app ID and object ID are securely stored for future reference.

Each tenant supports a defined quota of blueprints, and each blueprint can have up to a defined number of child identities. If your use case requires additional blueprints or identities beyond the default limits, contact Microsoft support to request a quota increase. While duplicate display names are permitted, using unique names for each blueprint is recommended for improved manageability and traceability.

2.2 Sample PowerShell Script for Listing Agent Blueprints

`scripts/ListAgentBlueprints.ps1` is a sample script that demonstrates how to list all agent blueprints in *Azure Entra ID* using PowerShell. It retrieves blueprints from the Microsoft Graph beta endpoint and displays them as formatted JSON output. It serves as a reference implementation for enumerating agent blueprints, which can be further automated in the future.

2.3 Sample PowerShell Script for Deleting Agent Blueprints

scripts/DeleteAgentBlueprints.ps1 is a sample script that demonstrates how to delete agent blueprints by display name in *Azure Entra ID* using PowerShell. The `$blueprint_name` parameter specifies the display name of the blueprints to delete and can be passed as a script parameter or set as a session variable. The script authenticates to Microsoft Graph with the necessary permissions, retrieves all blueprints from the Microsoft Graph beta endpoint, filters those matching the specified display name, and issues DELETE requests to remove each matching blueprint. It serves as a reference implementation for deprovisioning agent blueprints, which can be further automated in the future.

3 Azure Agent Identity

In *Azure Entra ID*, an *agent identity* is a first-class security principal assigned to an autonomous agent—a software entity that performs tasks and makes decisions without human intervention. This identity enables the agent to authenticate, obtain authorization tokens, and interact with resources across the Azure ecosystem. Agent identities are foundational to ensuring that autonomous agents operate under well-defined permissions while adhering to organizational security and compliance policies.

Each agent identity is derived from a parent *agent blueprint* (see §2) that defines the permissions, capabilities, and credential management strategy for all child identities. This hierarchical relationship enables centralized governance, allowing organizations to enforce consistent security policies across all agents derived from the same blueprint. By leveraging agent identities, organizations can implement the principle of least privilege, granting agents only the access necessary to perform their designated tasks and reducing the overall attack surface.

Each agent identity is associated with a unique identifier that serves as the basis for permission management and access control. Organizations can leverage this identifier to enforce fine-grained authorization policies, restricting agents to the minimum set of resources required for their designated tasks. Additionally, agent identities provide a durable audit trail, enabling organizations to trace agent actions, analyze behavioral patterns, and maintain accountability throughout the system lifecycle.

Currently, creating and managing agent identities in *Azure Entra ID* requires familiarity with *PowerShell* and the *Microsoft Graph PowerShell SDK*. However, we envision a future state where agent identities can be programmatically created, managed, and rotated at scale, with seamless integration into the agent development lifecycle and DevOps pipelines. This would empower organizations to rapidly onboard new agents, enforce security best practices automatically, and maintain robust governance as the number of agents grows.

3.1 Sample PowerShell Script for Creating an Agent Identity

scripts/CreateAgentIdentity.ps1 is a sample script that demonstrates how to create an agent identity in *Azure Entra ID* using PowerShell. The script requires the following parameters: `$tenant_id` (the Azure tenant ID), `$agent_id_name` (the display name for the new identity), `$blueprint_app_id` (the app ID of the parent blueprint), `$client_secret` (the blueprint's client secret for token acquisition), and `$sponsor_user_id` (the object ID of the sponsoring user). These can be passed as script parameters or set as session variables. The script authenticates to Microsoft Graph with the necessary permissions, acquires a blueprint token via client credentials, checks for existing identities with the same name, and issues a POST request to the Microsoft Graph beta endpoint to create the agent identity. It serves as a reference implementation for provisioning agent identities, which can be further automated in the future.

Each blueprint supports a defined quota of child agent identities. If your use case requires additional identities beyond the default limit, contact Microsoft support to request a quota increase. While duplicate display names are permitted, using unique names for each agent identity is recommended for improved manageability and traceability.

After the script creates the agent identity, the entry may not appear immediately in the list of identities due

to eventual consistency in the system. However, the identity is functional and can be used for authentication and authorization as soon as the creation API call returns successfully. The app ID and object ID displayed in the console output can be used immediately, and the identity will become visible in subsequent list queries after a brief propagation delay.

3.2 Sample PowerShell Script for Listing Agent Identities

`scripts/ListAgentIdentities.ps1` is a sample script that demonstrates how to list all agent identities in *Azure Entra ID* using PowerShell. It retrieves agent identities from the Microsoft Graph beta endpoint and displays them as formatted JSON output. It serves as a reference implementation for enumerating agent identities, which can be further automated in the future.

3.3 Sample PowerShell Script for Deleting Agent Identities

`scripts/DeleteAgentIdentities.ps1` is a sample script that demonstrates how to delete agent identities by display name in *Azure Entra ID* using PowerShell. The `$agent_id_name` parameter specifies the display name of the identities to delete and can be passed as a script parameter or set as a session variable. The script retrieves all agent identities from the Microsoft Graph beta endpoint, filters those matching the specified display name, and issues DELETE requests to remove each matching identity. It serves as a reference implementation for deprovisioning agent identities, which can be further automated in the future.

4 Managed Identity

In *Azure Entra ID*, a *managed identity* is a type of service principal that provides an automatically managed identity for applications to use when connecting to resources that support Microsoft Entra authentication. Managed identities eliminate the need for developers to manage credentials, as Azure automatically handles the creation, rotation, and security of the identity. There are two types of managed identities:

- *System-assigned managed identity*: This identity is enabled directly on an Azure resource, such as a virtual machine or an Azure Function. It is tied to the lifecycle of that resource, meaning it is created when the resource is created and deleted when the resource is deleted.
- *User-assigned managed identity*: This identity is created as a standalone Azure resource and can be assigned to one or more Azure resources. It has its own lifecycle, independent of the resources it is assigned to, allowing for greater flexibility and reuse across multiple resources.

In this context, we create *user-assigned managed identities* that are associated with an agent blueprint (see §2) and serve as the underlying identity for agent instances. By leveraging managed identities, organizations can ensure that agent identities are securely managed by Azure, with automatic credential rotation and reduced risk of credential leakage. This approach also simplifies the authentication process for agents, enabling them to seamlessly obtain access tokens for Azure resources without handling secrets directly.

4.1 Sample PowerShell Script for Creating a User-Assigned Managed Identity

`scripts/CreateUserManagedIdentity.ps1` is a sample script that demonstrates how to create a user-assigned managed identity in *Azure Entra ID* using PowerShell. The script requires the following parameters: `$subscription_id` (the Azure subscription ID), `$mi_name` (the name for the new managed identity), `$rg_name` (the resource group in which the managed identity will be created), and `$location` (the Azure region, defaulting to `eastus2`). These can be passed as script parameters or set as session variables. The script verifies that the current user has the required role assignment on the resource group, checks for an existing managed identity with the same name, and creates a new user-assigned managed identity using the `New-AzUserAssignedIdentity` cmdlet if one does not already exist. The script outputs the details of the created managed identity for future reference and association with agent blueprints. This step can also be performed in the Azure portal.

4.2 Sample PowerShell Script for Deleting a User-Assigned Managed Identity

scripts/DeleteUserManagedIdentity.ps1 is a sample script that demonstrates how to delete a user-assigned managed identity in *Azure Entra ID* using PowerShell. The script requires the following parameters: `$rg_name` (the resource group containing the managed identity) and `$mi_name` (the name of the managed identity to delete). These can be passed as script parameters or set as session variables. The script checks for the existence of the specified managed identity, and if it exists, deletes it using the `Remove-AzUserAssignedIdentity` cmdlet. Upon successful deletion, the script outputs a confirmation message. This step can also be performed in the Azure portal.

5 Federated Identity Credential

A *federated identity credential* in *Azure Entra ID* enables an application or service to authenticate using external identities—such as those issued by GitHub Actions or Azure Managed Identities—without traditional secrets or certificates. This credential type leverages the OpenID Connect (OIDC) protocol to establish a trust relationship between *Azure Entra ID* and the external identity provider, enabling secure, passwordless authentication. Adopting federated identity credentials reduces the attack surface associated with credential management, strengthens overall security posture, and streamlines integration with modern development and deployment workflows.

Each federated identity credential is associated with an *agent blueprint* (§2) and serves as the authentication mechanism for agent instances that access resources across the Azure ecosystem. Because Azure manages the credential lifecycle—including automatic rotation—the risk of credential leakage is significantly reduced. This approach simplifies the authentication process for agents, enabling them to obtain access tokens for Azure resources without handling secrets directly.

5.1 Sample PowerShell Script for Creating a Federated Identity Credential

scripts/CreateFederatedIdentityCredential.ps1 is a reference script that creates or updates a federated identity credential on an *agent blueprint* in *Azure Entra ID*. The script accepts the following parameters: `$blueprint_app_id` (the app ID of the parent blueprint), `$subject` (the subject claim, such as a managed identity principal ID or a GitHub Actions reference), `$issuer` (the issuer URL of the external identity provider), `$audiences` (the token audience; defaults to `api://AzureADTokenExchange`), `$description` (an optional description), and `$fed_cred_name` (the credential name). The script authenticates to Microsoft Graph with the required permissions, verifies that the specified blueprint exists, checks for an existing credential with the same name, prompts for confirmation if the subject differs, and issues a POST or PATCH request to the Microsoft Graph beta endpoint accordingly. This implementation can be adapted or automated to suit organizational provisioning workflows.

5.2 Sample PowerShell Script for Listing Federated Identity Credentials

scripts/ListFederatedIdentityCredentials.ps1 is a reference script that lists the federated identity credentials associated with a specific *agent blueprint* in *Azure Entra ID*. The script accepts the parameter `$blueprint_app_id` (the app ID of the parent blueprint). It authenticates to Microsoft Graph with the required permissions, retrieves all federated identity credentials for the specified blueprint, and outputs each credential's name, subject claim, issuer, audiences, and description. This capability supports auditing and governance by providing visibility into the credentials used for agent authentication.

5.3 Sample PowerShell Script for Deleting a Federated Identity Credential

scripts/DeleteFederatedIdentityCredential.ps1 is a reference script that deletes a federated identity credential from an *agent blueprint* in *Azure Entra ID*. The script accepts the parameters `$blueprint_app_id` (the app ID of the parent blueprint) and `$subject` (the subject claim of the credential to remove). It

authenticates to Microsoft Graph with the required permissions, retrieves the federated identity credentials for the specified blueprint, identifies the credential matching the provided subject claim, and issues a DELETE request to the Microsoft Graph beta endpoint. Timely removal of unused or compromised credentials is essential for maintaining a sound security posture. This implementation can be adapted or automated to suit organizational deprovisioning workflows.

6 Configure Identifier URI and Scopes for an Agent Blueprint

Configuring the identifier URI and scopes is essential for securing agent identity. The identifier URI establishes a unique endpoint that enables the agent to receive authorized, token-based requests on behalf of users. Scopes define granular permissions for API operations, which the backend validates from incoming access tokens.

- *Secure Authentication and Authorization*: The identifier URI creates a unique identifier for the agent, enabling it to act on behalf of users in interactive scenarios.
- *Defining Access Permissions (Scopes)*: Scopes specify what an agent is permitted to do, serving as granular permissions validated by the backend from incoming access tokens.

6.1 Reference Script for Configuring Identifier URI and Scopes

`scripts/ConfigureIdentifierURIandScopes.ps1` is a reference script for configuring the identifier URI and API scopes on an agent blueprint in *Azure Entra ID*. The script accepts the following parameters: `$tenant_id` (the Azure tenant ID), `$blueprint_app_id` (the app ID of the agent blueprint), `$identifier_uri` (the unique identifier URI; defaults to `api://$blueprint_app_id` when omitted), `$scope_name` (the scope name, e.g., `read.all`), `$scope_desc` (a description for the scope), `$scope_display_name` (the display name for the scope), and `$scope_type` (either `User` or `Admin`; defaults to `User`). Each parameter can be passed directly or set as a session variable. The script authenticates to Microsoft Graph, constructs the request body with the specified identifier URI and scopes, and issues a PATCH request to the beta endpoint to update the blueprint's application registration.

6.2 Reference Script for Listing Identifier URI and Scopes

`scripts/ListIdentifierURIandScopes.ps1` is a reference script for listing the identifier URI and API scopes configured on an agent blueprint in *Azure Entra ID*. The script accepts the `$tenant_id` and `$blueprint_app_id` parameters, authenticates to Microsoft Graph, retrieves the application registration for the specified blueprint, and outputs the identifier URIs and OAuth2 permission scopes.

6.3 Reference Script for Removing an Identifier URI

`scripts/RemoveIdentifierURI.ps1` is a reference script for removing an identifier URI from an agent blueprint in *Azure Entra ID*. The script accepts the following parameters: `$tenant_id` (the Azure tenant ID), `$blueprint_app_id` (the app ID of the agent blueprint), and `$identifier_uri` (the identifier URI to remove). Each parameter can be passed directly or set as a session variable. The script authenticates to Microsoft Graph, retrieves the current identifier URIs, filters out the specified URI, and issues a PATCH request to the beta endpoint to update the blueprint.

6.4 Reference Script for Removing a Scope

`scripts/RemoveScope.ps1` is a reference script for removing an API scope from an agent blueprint in *Azure Entra ID*. The script accepts the following parameters: `$tenant_id` (the Azure tenant ID), `$blueprint_app_id` (the app ID of the agent blueprint), and `$scope_id` (the ID of the scope to remove). Each parameter can be passed directly or set as a session variable. The script authenticates to Microsoft Graph, disables the

target scope, then removes it from the blueprint via two sequential PATCH requests to the beta endpoint, as required by the Graph API.

7 Grant Storage Blob Data Reader Role to an Agent Identity

To enable an agent identity to read blob data in Azure Storage, it must be granted the *Storage Blob Data Reader* role. This role provides read access to blob containers and their contents. It can be assigned at the resource group or individual storage account scope, depending on the level of access required.

7.1 Reference Script for Assigning the Storage Blob Data Reader Role

scripts/SetupStoragePermissions.ps1 is a reference script for assigning the *Storage Blob Data Reader* role to an agent identity on a specific Azure Storage account. The script accepts three parameters: `$storage_acc_name` (the storage account name), `$rg_name` (the resource group containing the storage account), and `$agent_id_id` (the application ID of the agent identity). Each parameter can be passed directly or set as a session variable. The script validates the storage account, checks for an existing role assignment, and uses the Azure CLI (`az role assignment create`) to grant the role at the storage account scope. Alternatively, this can be performed in the Azure portal by navigating to the storage account, selecting "Access Control (IAM)", and adding a role assignment for the agent identity. The user performing this action must have sufficient permissions, such as *Owner*, *User Access Administrator*, or *Role Based Access Control Administrator* on the resource group or storage account.

7.2 Reference Script for Removing the Storage Blob Data Reader Role

scripts/RemoveStoragePermission.ps1 is a reference script for removing the *Storage Blob Data Reader* role assignment from an agent identity on a specific Azure Storage account. The script accepts the same three parameters: `$storage_acc_name`, `$rg_name`, and `$agent_id_id`. It validates the storage account, confirms the role assignment exists, and uses the Azure CLI (`az role assignment delete`) to remove it. Alternatively, this can be performed in the Azure portal by navigating to the storage account, selecting "Access Control (IAM)", and removing the role assignment for the agent identity. The user performing this action must have sufficient permissions, such as *Owner*, *User Access Administrator*, or *Role Based Access Control Administrator* on the resource group or storage account.