# Azure Identity client library for .NET

The Azure Identity library provides [Microsoft Entra ID](https://learn.microsoft.com/entra/fundamentals/whatis) ([formerly Azure Active Directory](https://learn.microsoft.com/entra/fundamentals/new-name)) token authentication support across the Azure SDK. It provides a set of [TokenCredential](https://learn.microsoft.com/dotnet/api/azure.core.tokencredential?view=azure-dotnet) implementations which can be used to construct Azure SDK clients which support Microsoft Entra token authentication.

[Source code](https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/identity/Azure.Identity/src) | [Package (NuGet)](https://www.nuget.org/packages/Azure.Identity) | [API reference documentation](https://learn.microsoft.com/dotnet/api/azure.identity?view=azure-dotnet) | [Microsoft Entra ID documentation](https://learn.microsoft.com/entra/identity/)

## Getting started

### Install the package

Install the Azure Identity client library for .NET with [NuGet](https://www.nuget.org/):

dotnet add package Azure.Identity

### Prerequisites

* An [Azure subscription](https://azure.microsoft.com/free/dotnet/).
* The [Azure CLI](https://learn.microsoft.com/cli/azure) can also be useful for authenticating in a development environment, creating accounts, and managing account roles.

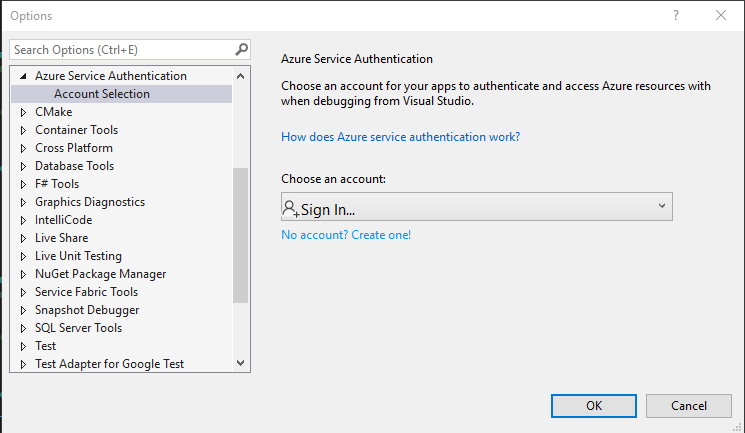
### Authenticate the client

When debugging and executing code locally it is typical for a developer to use their own account for authenticating calls to Azure services. There are several developer tools which can be used to perform this authentication in your development environment.

#### Authenticate via Visual Studio

Developers using Visual Studio 2017 or later can authenticate a Microsoft Entra account through the IDE. Applications using the DefaultAzureCredential or the VisualStudioCredential can then use this account to authenticate calls in their application when running locally.

To authenticate in Visual Studio, select the **Tools** > **Options** menu to launch the Options dialog. Then navigate to the Azure Service Authentication options to sign in with your Microsoft Entra account.



#### Authenticate via Visual Studio Code

Developers using Visual Studio Code can use the [Azure Account extension](https://marketplace.visualstudio.com/items?itemName=ms-vscode.azure-account) to authenticate via the editor. Applications using the DefaultAzureCredential or the VisualStudioCodeCredential can then use this account to authenticate calls in their application when running locally.

It's a [known issue](https://github.com/Azure/azure-sdk-for-net/issues/30525) that VisualStudioCodeCredential doesn't work with [Azure Account extension](https://marketplace.visualstudio.com/items?itemName=ms-vscode.azure-account) versions newer than **0.9.11**. A long-term fix to this problem is in progress. In the meantime, consider [authenticating via the Azure CLI](#authenticating-via-the-azure-cli).

#### Authenticate via the Azure CLI

Developers coding outside of an IDE can also use the [Azure CLI](https://learn.microsoft.com/cli/azure) to authenticate. Applications using the DefaultAzureCredential or the AzureCliCredential can then use this account to authenticate calls in their application when running locally.

To authenticate with the [Azure CLI](https://learn.microsoft.com/cli/azure), users can run the command az login. For users running on a system with a default web browser, the Azure CLI will launch the browser to authenticate the user.

Azure CLI Account Sign In

For systems without a default web browser, the az login command will use the device code authentication flow. The user can also force the Azure CLI to use the device code flow rather than launching a browser by specifying the --use-device-code argument.

Azure CLI Account Device Code Sign In

#### Authenticate via the Azure Developer CLI

Developers coding outside of an IDE can also use the [Azure Developer CLI](https://aka.ms/azure-dev) to authenticate. Applications using the DefaultAzureCredential or the AzureDeveloperCliCredential can then use this account to authenticate calls in their application when running locally.

To authenticate with the [Azure Developer CLI](https://aka.ms/azure-dev), users can run the command azd auth login. For users running on a system with a default web browser, the Azure Developer CLI will launch the browser to authenticate the user.

For systems without a default web browser, the azd auth login --use-device-code command will use the device code authentication flow.

#### Authenticate via Azure PowerShell

Developers coding outside of an IDE can also use [Azure PowerShell](https://learn.microsoft.com/powershell/azure) to authenticate. Applications using the DefaultAzureCredential or the AzurePowerShellCredential can then use this account to authenticate calls in their application when running locally.

To authenticate with [Azure PowerShell](https://learn.microsoft.com/powershell/azure), users can run the command Connect-AzAccount. For users running on a system with a default web browser and version 5.0.0 or later of azure PowerShell, it will launch the browser to authenticate the user.

For systems without a default web browser, the Connect-AzAccount command will use the device code authentication flow. The user can also force Azure PowerShell to use the device code flow rather than launching a browser by specifying the UseDeviceAuthentication argument.

## Key concepts

### Credentials

A credential is a class which contains or can obtain the data needed for a service client to authenticate requests. Service clients across the Azure SDK accept credentials when they're constructed. Service clients use those credentials to authenticate requests to the service.

The Azure Identity library focuses on OAuth authentication with Microsoft Entra ID, and it offers a variety of credential classes capable of acquiring a Microsoft Entra token to authenticate service requests. All of the credential classes in this library are implementations of the TokenCredential abstract class in [Azure.Core](https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/core/Azure.Core), and any of them can be used to construct service clients capable of authenticating with a TokenCredential.

See [Credential Classes](#credential-classes) for a complete listing of available credential types.

### DefaultAzureCredential

The DefaultAzureCredential is appropriate for most scenarios where the application is intended to ultimately be run in Azure. This is because the DefaultAzureCredential combines credentials commonly used to authenticate when deployed, with credentials used to authenticate in a development environment.

Note: DefaultAzureCredential is intended to simplify getting started with the SDK by handling common scenarios with reasonable default behaviors. Developers who want more control or whose scenario isn't served by the default settings should use other credential types.

The DefaultAzureCredential attempts to authenticate via the following mechanisms, in this order, stopping when one succeeds:

DefaultAzureCredential authentication flow

1. **Environment** - The DefaultAzureCredential will read account information specified via [environment variables](#environment-variables) and use it to authenticate.
2. **Workload Identity** - If the application is deployed to an Azure host with Workload Identity enabled, the DefaultAzureCredential will authenticate with that account.
3. **Managed Identity** - If the application is deployed to an Azure host with Managed Identity enabled, the DefaultAzureCredential will authenticate with that account.
4. **Visual Studio** - If the developer has authenticated via Visual Studio, the DefaultAzureCredential will authenticate with that account.
5. **Visual Studio Code** - Currently excluded by default as SDK authentication via Visual Studio Code is broken due to issue [#27263](https://github.com/Azure/azure-sdk-for-net/issues/27263). The VisualStudioCodeCredential will be re-enabled in the DefaultAzureCredential flow once a fix is in place. Issue [#30525](https://github.com/Azure/azure-sdk-for-net/issues/30525) tracks this. In the meantime Visual Studio Code users can authenticate their development environment using the [Azure CLI](https://learn.microsoft.com/cli/azure/).
6. **Azure CLI** - If the developer has authenticated an account via the Azure CLI az login command, the DefaultAzureCredential will authenticate with that account.
7. **Azure PowerShell** - If the developer has authenticated an account via the Azure PowerShell Connect-AzAccount command, the DefaultAzureCredential will authenticate with that account.
8. **Azure Developer CLI** - If the developer has authenticated via the Azure Developer CLI azd auth login command, the DefaultAzureCredential will authenticate with that account.
9. **Interactive browser** - If enabled, the DefaultAzureCredential will interactively authenticate the developer via the current system's default browser. By default, this credential type is disabled.

#### Continuation policy

As of version 1.10.1, DefaultAzureCredential will attempt to authenticate with all developer credentials until one succeeds, regardless of any errors previous developer credentials experienced. For example, a developer credential may attempt to get a token and fail, so DefaultAzureCredential will continue to the next credential in the flow. Deployed service credentials will stop the flow with a thrown exception if they're able to attempt token retrieval, but don't receive one. Prior to version 1.10.1, developer credentials would similarly stop the authentication flow if token retrieval failed.

This behavior allows for trying all of the developer credentials on your machine while having predictable deployed behavior.

## Examples

### Authenticate with DefaultAzureCredential

This example demonstrates authenticating the SecretClient from the [Azure.Security.KeyVault.Secrets](https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/keyvault/Azure.Security.KeyVault.Secrets) client library using the DefaultAzureCredential.

// Create a secret client using the DefaultAzureCredential

var client = new SecretClient(new Uri("https://myvault.vault.azure.net/"), new DefaultAzureCredential());

### Enable interactive authentication with DefaultAzureCredential

Interactive authentication is disabled in the DefaultAzureCredential by default. This example demonstrates two ways of enabling the interactive authentication portion of the DefaultAzureCredential. When enabled the DefaultAzureCredential will fall back to interactively authenticating the developer via the system's default browser if when no other credentials are available. This example then authenticates an EventHubProducerClient from the [Azure.Messaging.EventHubs](https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/eventhub/Azure.Messaging.EventHubs) client library using the DefaultAzureCredential with interactive authentication enabled.

// the includeInteractiveCredentials constructor parameter can be used to enable interactive authentication

var credential = new DefaultAzureCredential(includeInteractiveCredentials: true);

var eventHubClient = new EventHubProducerClient("myeventhub.eventhubs.windows.net", "myhubpath", credential);

### Specify a user-assigned managed identity with DefaultAzureCredential

Many Azure hosts allow the assignment of a user-assigned managed identity. This example demonstrates configuring the DefaultAzureCredential to authenticate a user-assigned identity when deployed to an Azure host. It then authenticates a BlobClient from the [Azure.Storage.Blobs](https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/storage/Azure.Storage.Blobs) client library with credential.

// When deployed to an azure host, the default azure credential will authenticate the specified user assigned managed identity.

string userAssignedClientId = "<your managed identity client Id>";

var credential = new DefaultAzureCredential(new DefaultAzureCredentialOptions { ManagedIdentityClientId = userAssignedClientId });

var blobClient = new BlobClient(new Uri("https://myaccount.blob.core.windows.net/mycontainer/myblob"), credential);

In addition to configuring the ManagedIdentityClientId via code, it can also be set using the AZURE\_CLIENT\_ID environment variable. These two approaches are equivalent when using the DefaultAzureCredential.

### Define a custom authentication flow with ChainedTokenCredential

While the DefaultAzureCredential is generally the quickest way to get started developing applications for Azure, more advanced users may want to customize the credentials considered when authenticating. The ChainedTokenCredential enables users to combine multiple credential instances to define a customized chain of credentials. This example demonstrates creating a ChainedTokenCredential which will attempt to authenticate using managed identity, and fall back to authenticating via the Azure CLI if managed identity is unavailable in the current environment. The credential is then used to authenticate an EventHubProducerClient from the [Azure.Messaging.EventHubs](https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/eventhub/Azure.Messaging.EventHubs) client library.

// Authenticate using managed identity if it is available; otherwise use the Azure CLI to authenticate.

var credential = new ChainedTokenCredential(new ManagedIdentityCredential(), new AzureCliCredential());

var eventHubProducerClient = new EventHubProducerClient("myeventhub.eventhubs.windows.net", "myhubpath", credential);

## Managed identity support

[Managed identity authentication](https://learn.microsoft.com/entra/identity/managed-identities-azure-resources/overview) is supported via either the DefaultAzureCredential or the ManagedIdentityCredential directly for the following Azure services:

* [Azure App Service and Azure Functions](https://learn.microsoft.com/azure/app-service/overview-managed-identity?tabs=dotnet)
* [Azure Arc](https://learn.microsoft.com/azure/azure-arc/servers/managed-identity-authentication)
* [Azure Cloud Shell](https://learn.microsoft.com/azure/cloud-shell/msi-authorization)
* [Azure Kubernetes Service](https://learn.microsoft.com/azure/aks/use-managed-identity)
* [Azure Service Fabric](https://learn.microsoft.com/azure/service-fabric/concepts-managed-identity)
* [Azure Virtual Machines](https://learn.microsoft.com/entra/identity/managed-identities-azure-resources/how-to-use-vm-token)
* [Azure Virtual Machines Scale Sets](https://learn.microsoft.com/entra/identity/managed-identities-azure-resources/qs-configure-powershell-windows-vmss)

### Examples

These examples demonstrate authenticating the SecretClient from the [Azure.Security.KeyVault.Secrets](https://github.com/Azure/azure-sdk-for-net/tree/main/sdk/keyvault/Azure.Security.KeyVault.Secrets) client library using the ManagedIdentityCredential.

#### Authenticate with a user-assigned managed identity

var credential = new ManagedIdentityCredential(clientId: userAssignedClientId);

var client = new SecretClient(new Uri("https://myvault.vault.azure.net/"), credential);

#### Authenticate with a system-assigned managed identity

var credential = new ManagedIdentityCredential();

var client = new SecretClient(new Uri("https://myvault.vault.azure.net/"), credential);

## Cloud configuration

Credentials default to authenticating to the Microsoft Entra endpoint for the Azure public cloud. To access resources in other clouds, such as Azure Government or a private cloud, configure credentials with the AuthorityHost argument. [AzureAuthorityHosts](https://learn.microsoft.com/dotnet/api/azure.identity.azureauthorityhosts?view=azure-dotnet) defines authorities for well-known clouds:

var credential = new DefaultAzureCredential(new DefaultAzureCredentialOptions { AuthorityHost = AzureAuthorityHosts.AzureGovernment });

Not all credentials require this configuration. Credentials which authenticate through a development tool, such as AzureCliCredential, use that tool's configuration.

## Credential classes

### Authenticate Azure-hosted applications

| **Credential** | **Usage** |
| --- | --- |
| [DefaultAzureCredential](https://learn.microsoft.com/dotnet/api/azure.identity.defaultazurecredential?view=azure-dotnet) | Provides a simplified authentication experience to quickly start developing applications run in Azure. |
| [ChainedTokenCredential](https://learn.microsoft.com/dotnet/api/azure.identity.chainedtokencredential?view=azure-dotnet) | Allows users to define custom authentication flows composing multiple credentials. |
| [EnvironmentCredential](https://learn.microsoft.com/dotnet/api/azure.identity.environmentcredential?view=azure-dotnet) | Authenticates a service principal or user via credential information specified in environment variables. |
| [ManagedIdentityCredential](https://learn.microsoft.com/dotnet/api/azure.identity.managedidentitycredential?view=azure-dotnet) | Authenticates the managed identity of an Azure resource. |
| [WorkloadIdentityCredential](https://learn.microsoft.com/dotnet/api/azure.identity.workloadidentitycredential?view=azure-dotnet) | Supports [Microsoft Entra Workload ID](https://learn.microsoft.com/azure/aks/workload-identity-overview) on Kubernetes. |

### Authenticate service principals

| **Credential** | **Usage** | **Reference** |
| --- | --- | --- |
| [ClientAssertionCredential](https://learn.microsoft.com/dotnet/api/azure.identity.clientassertioncredential?view=azure-dotnet) | Authenticates a service principal using a signed client assertion. |  |
| [ClientCertificateCredential](https://learn.microsoft.com/dotnet/api/azure.identity.clientcertificatecredential?view=azure-dotnet) | Authenticates a service principal using a certificate. | [Service principal authentication](https://learn.microsoft.com/entra/identity-platform/app-objects-and-service-principals) |
| [ClientSecretCredential](https://learn.microsoft.com/dotnet/api/azure.identity.clientsecretcredential?view=azure-dotnet) | Authenticates a service principal using a secret. | [Service principal authentication](https://learn.microsoft.com/entra/identity-platform/app-objects-and-service-principals) |

### Authenticate users

| **Credential** | **Usage** | **Reference** |
| --- | --- | --- |
| [AuthorizationCodeCredential](https://learn.microsoft.com/dotnet/api/azure.identity.authorizationcodecredential?view=azure-dotnet) | Authenticates a user with a previously obtained authorization code. | [OAuth2 authentication code](https://learn.microsoft.com/entra/identity-platform/v2-oauth2-auth-code-flow) |
| [DeviceCodeCredential](https://learn.microsoft.com/dotnet/api/azure.identity.devicecodecredential?view=azure-dotnet) | Interactively authenticates a user on devices with limited UI. | [Device code authentication](https://learn.microsoft.com/entra/identity-platform/v2-oauth2-device-code) |
| [InteractiveBrowserCredential](https://learn.microsoft.com/dotnet/api/azure.identity.interactivebrowsercredential?view=azure-dotnet) | Interactively authenticates a user with the default system browser. | [OAuth2 authentication code](https://learn.microsoft.com/entra/identity-platform/v2-oauth2-auth-code-flow) |
| [OnBehalfOfCredential](https://learn.microsoft.com/dotnet/api/azure.identity.onbehalfofcredential?view=azure-dotnet) | Propagates the delegated user identity and permissions through the request chain. | [On-behalf-of authentication](https://learn.microsoft.com/entra/identity-platform/v2-oauth2-on-behalf-of-flow) |
| [UsernamePasswordCredential](https://learn.microsoft.com/dotnet/api/azure.identity.usernamepasswordcredential?view=azure-dotnet) | Authenticates a user with a username and password. | [Username + password authentication](https://learn.microsoft.com/entra/identity-platform/v2-oauth-ropc) |

### Authenticate via development tools

| **Credential** | **Usage** | **Reference** |
| --- | --- | --- |
| [AzureCliCredential](https://learn.microsoft.com/dotnet/api/azure.identity.azureclicredential?view=azure-dotnet) | Authenticates in a development environment with the Azure CLI. | [Azure CLI authentication](https://learn.microsoft.com/cli/azure/authenticate-azure-cli) |
| [AzureDeveloperCliCredential](https://learn.microsoft.com/dotnet/api/azure.identity.azuredeveloperclicredential?view=azure-dotnet) | Authenticates in a development environment with the Azure Developer CLI. | [Azure Developer CLI Reference](https://learn.microsoft.com/azure/developer/azure-developer-cli/reference) |
| [AzurePowerShellCredential](https://learn.microsoft.com/dotnet/api/azure.identity.azurepowershellcredential?view=azure-dotnet) | Authenticates in a development environment with the Azure PowerShell. | [Azure PowerShell authentication](https://learn.microsoft.com/powershell/azure/authenticate-azureps) |
| [VisualStudioCredential](https://learn.microsoft.com/dotnet/api/azure.identity.visualstudiocredential?view=azure-dotnet) | Authenticates in a development environment with Visual Studio. | [Visual Studio configuration](https://learn.microsoft.com/dotnet/azure/configure-visual-studio) |
| [VisualStudioCodeCredential](https://learn.microsoft.com/dotnet/api/azure.identity.visualstudiocodecredential?view=azure-dotnet) | Authenticates as the user signed in to the Visual Studio Code Azure Account extension. | [VS Code Azure Account extension](https://marketplace.visualstudio.com/items?itemName=ms-vscode.azure-account) |

**Note:** All credential implementations in the Azure Identity library are threadsafe, and a single credential instance can be used by multiple service clients.

## Environment variables

[DefaultAzureCredential](https://learn.microsoft.com/dotnet/api/azure.identity.defaultazurecredential?view=azure-dotnet) and [EnvironmentCredential](https://learn.microsoft.com/dotnet/api/azure.identity.environmentcredential?view=azure-dotnet) can be configured with environment variables. Each type of authentication requires values for specific variables:

### Service principal with secret

| **Variable name** | **Value** |
| --- | --- |
| AZURE\_CLIENT\_ID | ID of a Microsoft Entra application |
| AZURE\_TENANT\_ID | ID of the application's Microsoft Entra tenant |
| AZURE\_CLIENT\_SECRET | one of the application's client secrets |

### Service principal with certificate

| **variable name** | **Value** |
| --- | --- |
| AZURE\_CLIENT\_ID | ID of a Microsoft Entra application |
| AZURE\_TENANT\_ID | ID of the application's Microsoft Entra tenant |
| AZURE\_CLIENT\_CERTIFICATE\_PATH | path to a PFX or PEM-encoded certificate file including private key |
| AZURE\_CLIENT\_CERTIFICATE\_PASSWORD | (optional) the password protecting the certificate file (currently only supported for PFX (PKCS12) certificates) |
| AZURE\_CLIENT\_SEND\_CERTIFICATE\_CHAIN | (optional) send certificate chain in x5c header to support subject name / issuer based authentication |

### Username and password

| **Variable name** | **Value** |
| --- | --- |
| AZURE\_CLIENT\_ID | ID of a Microsoft Entra application |
| AZURE\_TENANT\_ID | ID of the application's Microsoft Entra tenant |
| AZURE\_USERNAME | a username (usually an email address) |
| AZURE\_PASSWORD | that user's password |

### Managed identity (DefaultAzureCredential)

| **Variable name** | **Value** |
| --- | --- |
| AZURE\_CLIENT\_ID | The client ID for the user-assigned managed identity. If defined, used as the default value for ManagedIdentityClientId in DefaultAzureCredentialOptions |

Configuration is attempted in the above order. For example, if values for a client secret and certificate are both present, the client secret will be used.

## Continuous Access Evaluation

As of version 1.10.0, accessing resources protected by [Continuous Access Evaluation (CAE)](https://learn.microsoft.com/entra/identity/conditional-access/concept-continuous-access-evaluation) is possible on a per-request basis. This behavior can be enabled by setting the IsCaeEnabled property of TokenRequestContext via its constructor. CAE isn't supported for developer and managed identity credentials.

## Token caching

Token caching is a feature provided by the Azure Identity library that allows apps to:

* Cache tokens in memory (default) or on disk (opt-in).
* Improve resilience and performance.
* Reduce the number of requests made to Microsoft Entra ID to obtain access tokens.

The Azure Identity library offers both in-memory and persistent disk caching. For more details, see the [token caching documentation](https://github.com/Azure/azure-sdk-for-net/blob/main/sdk/identity/Azure.Identity/samples/TokenCache.md)

# Brokered Authentication

An authentication broker is an application that runs on a user’s machine and manages the authentication handshakes and token maintenance for connected accounts. Currently, only the Windows Web Account Manager (WAM) is supported. To enable support, use the Azure.Identity.Broker package. For details on authenticating using WAM, see the [broker package documentation](https://github.com/Azure/azure-sdk-for-net/blob/main/sdk/identity/Azure.Identity.Broker/README.md).

## Troubleshooting

See the [troubleshooting guide](https://github.com/Azure/azure-sdk-for-net/blob/main/sdk/identity/Azure.Identity/TROUBLESHOOTING.md) for details on how to diagnose various failure scenarios.

### Error handling

Errors arising from authentication can be raised on any service client method which makes a request to the service. This is because the first time the token is requested from the credential is on the first call to the service, and any subsequent calls might need to refresh the token. In order to distinguish these failures from failures in the service client Azure Identity classes raise the AuthenticationFailedException with details to the source of the error in the exception message as well as possibly the error message. Depending on the application these errors may or may not be recoverable.

using Azure.Identity;

using Azure.Security.KeyVault.Secrets;

// Create a secret client using the DefaultAzureCredential

var client = new SecretClient(new Uri("https://myvault.vault.azure.net/"), new DefaultAzureCredential());

try

{

KeyVaultSecret secret = await client.GetSecretAsync("secret1");

}

catch (AuthenticationFailedException e)

{

Console.WriteLine($"Authentication Failed. {e.Message}");

}

For more information on dealing with errors arising from failed requests to Microsoft Entra ID or managed identity endpoints, see the Microsoft Entra ID [documentation on authorization error codes](https://learn.microsoft.com/entra/identity-platform/reference-error-codes).

### Logging

The Azure Identity library provides the same [logging capabilities](https://github.com/Azure/azure-sdk-for-net/blob/main/sdk/core/Azure.Core/samples/Diagnostics.md#logging) as the rest of the Azure SDK.

The simplest way to see the logs to help debug authentication issues is to enable the console logging.

// Setup a listener to monitor logged events.

using AzureEventSourceListener listener = AzureEventSourceListener.CreateConsoleLogger();

All credentials can be configured with diagnostic options, in the same way as other clients in the SDK.

**CAUTION:** Requests and responses in the Azure Identity library contain sensitive information. Precaution must be taken to protect logs, when customizing the output, to avoid compromising account security.

DefaultAzureCredentialOptions options = new DefaultAzureCredentialOptions

{

Diagnostics =

{

LoggedHeaderNames = { "x-ms-request-id" },

LoggedQueryParameters = { "api-version" },

IsLoggingContentEnabled = true

}

};

When troubleshooting authentication issues, you may also want to enable logging of sensitive information. To enable this type of logging, set the IsLoggingContentEnabled property to true. To only log details about the account that was used to attempt authentication and authorization, set IsAccountIdentifierLoggingEnabled to true.

DefaultAzureCredentialOptions options = new DefaultAzureCredentialOptions

{

Diagnostics =

{

LoggedHeaderNames = { "x-ms-request-id" },

LoggedQueryParameters = { "api-version" },

IsAccountIdentifierLoggingEnabled = true

}

};

### Thread safety

We guarantee that all credential instance methods are thread-safe and independent of each other ([guideline](https://azure.github.io/azure-sdk/dotnet_introduction.html#dotnet-service-methods-thread-safety)). This ensures that the recommendation of reusing credential instances is always safe, even across threads.

### Additional concepts

[Client options](https://github.com/Azure/azure-sdk-for-net/blob/master/sdk/core/Azure.Core/README.md#configuring-service-clients-using-clientoptions) | [Accessing the response](https://github.com/Azure/azure-sdk-for-net/blob/master/sdk/core/Azure.Core/README.md#accessing-http-response-details-using-responset) | [Diagnostics](https://github.com/Azure/azure-sdk-for-net/blob/master/sdk/core/Azure.Core/samples/Diagnostics.md) | [Mocking](https://github.com/Azure/azure-sdk-for-net/blob/master/sdk/core/Azure.Core/README.md#mocking) | [Client lifetime](https://devblogs.microsoft.com/azure-sdk/lifetime-management-and-thread-safety-guarantees-of-azure-sdk-net-clients/)

## Next steps

### Client libraries supporting authentication with Azure Identity

Many of the client libraries listed [here](https://azure.github.io/azure-sdk/releases/latest/dotnet.html) support authenticating with TokenCredential and the Azure Identity library. There you will also find links where you can learn more about their use, including additional documentation and samples.

### Known Issues

This library doesn't currently support scenarios relating to the [Azure AD B2C](https://learn.microsoft.com/azure/active-directory-b2c/overview) service.

Open issues for the Azure.Identity library can be found [here](https://github.com/Azure/azure-sdk-for-net/issues?q=is%3Aissue+is%3Aopen+label%3AAzure.Identity).

## Contributing

This project welcomes contributions and suggestions. Most contributions require you to agree to a Contributor License Agreement (CLA) declaring that you have the right to, and actually do, grant us the rights to use your contribution. For details, visit <https://cla.microsoft.com>.

When you submit a pull request, a CLA-bot will automatically determine whether you need to provide a CLA and decorate the PR appropriately (e.g., label, comment). Simply follow the instructions provided by the bot. You will only need to do this once across all repos using our CLA.

This project has adopted the [Microsoft Open Source Code of Conduct](https://opensource.microsoft.com/codeofconduct/). For more information see the [Code of Conduct FAQ](https://opensource.microsoft.com/codeofconduct/faq/) or contact [opencode@microsoft.com](mailto:opencode@microsoft.com) with any additional questions or comments.