Configuring OpenID Connect in Azure

In this article

Use OpenID Connect within your workflows to authenticate with Azure.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

Overview

OpenID Connect (OIDC) allows your GitHub Actions workflows to access resources in Azure, without needing to store the Azure credentials as long-lived GitHub secrets.

This guide gives an overview of how to configure Azure to trust GitHub's OIDC as a federated identity, and includes a workflow example for the `azure/login` action that uses tokens to authenticate to Azure and access resources.

Prerequisites

- To learn the basic concepts of how GitHub uses OpenID Connect (OIDC), and its architecture and benefits, see "About security hardening with OpenID Connect."
- Before proceeding, you must plan your security strategy to ensure that access tokens are only allocated in a predictable way. To control how your cloud provider issues access tokens, you **must** define at least one condition, so that untrusted repositories can't request access tokens for your cloud resources. For more information, see "About security hardening with OpenID Connect."
- You must enable the following publicly accessible endpoints:
  - `https://HOSTNAME/_services/token/.well-known/openid-configuration`
  - `https://HOSTNAME/_services/token/.well-known/jwks`

  Note: Azure Active Directory (Azure AD) does not have fixed IP ranges defined for these endpoints.
- Make sure that the value of the issuer claim that's included with the JSON Web Token (JWT) is set to a publicly routable URL. For more information, see "About security hardening with OpenID Connect."

Adding the Federated Credentials to Azure

GitHub's OIDC provider works with Azure's workload identity federation. For an overview, see Microsoft's documentation at "Workload identity federation."

To configure the OIDC identity provider in Azure, you will need to perform the following configuration. For instructions on making these changes, refer to the Azure documentation.

1. Create an Azure Active Directory application and a service principal.
2. Add federated credentials for the Azure Active Directory application.
3. Create GitHub secrets for storing Azure configuration.

Additional guidance for configuring the identity provider:

- For security hardening, make sure you've reviewed "About security hardening with OpenID Connect." For an example, see "About security hardening with OpenID Connect."
- For the `audience` setting, `api://AzureADTokenExchange` is the recommended value, but you can also specify other values here.

Updating your GitHub Actions workflow

To update your workflows for OIDC, you will need to make two changes to your YAML:

1. Add permissions settings for the token.
2. Use the `azure/login` action to exchange the OIDC token (JWT) for a cloud access token.

Adding permissions settings

The job or workflow run requires a `permissions` setting with `id-token: write`. You won't be able to request the OIDC JWT ID token if the `permissions` setting for `id-token` is set to `read` or `none`.

The `id-token: write` setting allows the JWT to be requested from GitHub's OIDC provider using one of these approaches:

- Using environment variables on the runner (`ACTIONS_ID_TOKEN_REQUEST_URL` and `ACTIONS_ID_TOKEN_REQUEST_TOKEN`).
- Using `getIDToken()` from the Actions toolkit.

If you need to fetch an OIDC token for a workflow, then the permission can be set at the workflow level. For example:

YAML

```yaml
permissions:
  id-token: write # This is required for requesting the JWT
  contents: read  # This is required for actions/checkout
```

If you only need to fetch an OIDC token for a single job, then this permission can be set within that job. For example:

YAML

```yaml
permissions:
  id-token: write # This is required for requesting the JWT
```

You may need to specify additional permissions here, depending on your workflow's requirements.

For reusable workflows that are owned by the same user, organization, or enterprise as the caller workflow, the OIDC token generated in the reusable workflow can be accessed from the caller's context. For reusable workflows outside your enterprise or organization, the `permissions` setting for `id-token` should be explicitly set to `write` at the caller workflow level or in the specific job that calls the reusable workflow. This ensures that the OIDC token generated in the reusable workflow is only allowed to be consumed in the caller workflows when intended.

For more information, see "Reusing workflows."

Requesting the access token

The `azure/login` action receives a JWT from the GitHub OIDC provider, and then requests an access token from Azure. For more information, see the `azure/login` documentation.

The following example exchanges an OIDC ID token with Azure to receive an access token, which can then be used to access cloud resources.

YAML

```yaml
name: Run Azure Login with OIDC
on: [push]

permissions:
  id-token: write
  contents: read
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      - name: 'Az CLI login'
        uses: azure/login@v1
        with:
          client-id: ${{ secrets.AZURE_CLIENT_ID }}
          tenant-id: ${{ secrets.AZURE_TENANT_ID }}
          subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}

      - name: 'Run az commands'
        run: |
          az account show
          az group list
```

Further reading

- Using OpenID Connect with reusable workflows
- About self-hosted runners