About security hardening with OpenID Connect

In this article

OpenID Connect allows your workflows to exchange short-lived tokens directly from your cloud provider.

Overview of OpenID Connect

GitHub Actions workflows are often designed to access a cloud provider (such as AWS, Azure, GCP, or HashiCorp Vault) in order to deploy software or use the cloud's services. Before the workflow can access these resources, it will supply credentials, such as a password or token, to the cloud provider. These credentials are usually stored as a secret in GitHub, and the workflow presents this secret to the cloud provider every time it runs.

However, using hardcoded secrets requires you to create credentials in the cloud provider and then duplicate them in GitHub as a secret.

With OpenID Connect (OIDC), you can take a different approach by configuring your workflow to request a short-lived access token directly from the cloud provider. Your cloud provider also needs to support OIDC on their end, and you must configure a trust relationship that controls which workflows are able to request the access tokens. Providers that currently support OIDC include Amazon Web Services, Azure, Google Cloud Platform, and HashiCorp Vault, among others.

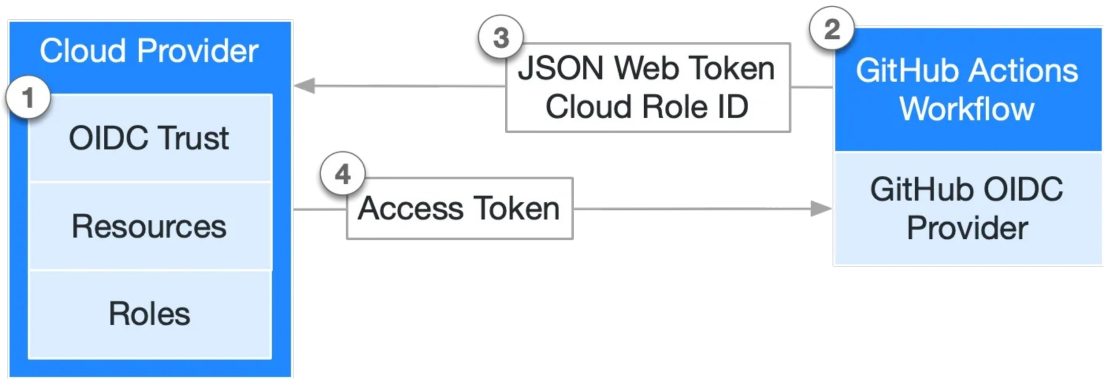
Benefits of using OIDC

By updating your workflows to use OIDC tokens, you can adopt the following good security practices:

- **No cloud secrets**: You won't need to duplicate your cloud credentials as long-lived GitHub secrets. Instead, you can configure the OIDC trust on your cloud provider, and then update your workflows to request a short-lived access token from the cloud provider through OIDC.
- **Authentication and authorization management**: You have more granular control over how workflows can use credentials, using your cloud provider's authentication (authN) and authorization (authZ) tools to control access to cloud resources.
- **Rotating credentials**: With OIDC, your cloud provider issues a short-lived access token that is only valid for a single job, and then automatically expires.

Getting started with OIDC

The following diagram gives an overview of how GitHub's OIDC provider integrates with your workflows and cloud provider:



1. In your cloud provider, create an OIDC trust between your cloud role and your GitHub workflow(s) that need access to the cloud.
2. Every time your job runs, GitHub's OIDC Provider auto-generates an OIDC token. This token contains multiple claims to establish a security-hardened and verifiable identity about the specific workflow that is trying to authenticate.
3. You could include a step or action in your job to request this token from GitHub's OIDC provider, and present it to the cloud provider.
4. Once the cloud provider successfully validates the claims presented in the token, it then provides a short-lived cloud access token that is available only for the duration of the job.

Configuring the OIDC trust with the cloud

When you configure your cloud to trust GitHub's OIDC provider, you **must** add conditions that filter incoming requests, so that untrusted repositories or workflows can’t request access tokens for your cloud resources:

- Before granting an access token, your cloud provider checks that the `subject` and other claims used to set conditions in its trust settings match those in the request's JSON Web Token (JWT). As a result, you must take care to correctly define the *subject* and other conditions in your cloud provider.
- The OIDC trust configuration steps and the syntax to set conditions for cloud roles (using *Subject* and other claims) will vary depending on which cloud provider you're using. For some examples, see "Example subject claims."

Understanding the OIDC token

Each job requests an OIDC token from GitHub's OIDC provider, which responds with an automatically generated JSON web token (JWT) that is unique for each workflow job where it is generated. When the job runs, the OIDC token is presented to the cloud provider. To validate the token, the cloud provider checks if the OIDC token's subject and other claims are a match for the conditions that were preconfigured on the cloud role's OIDC trust definition.

The following example OIDC token uses a subject (`sub`) that references a job environment named `prod` in the `octo-org/octo-repo` repository.

```
{
  "typ": "JWT",
  "alg": "RS256",
  "x5t": "example-thumbprint",
  "kid": "example-key-id"
}
{
  "jti": "example-id",
  "sub": "repo:octo-org/octo-repo:environment:prod",
  "environment": "prod",
  "aud": "https://github.com/octo-org",
  "ref": "refs/heads/main",
  "sha": "example-sha",
  "repository": "octo-org/octo-repo",
  "repository_owner": "octo-org",
  "actor_id": "12",
  "repository_visibility": "private",
  "repository_id": "74",
  "repository_owner_id": "65",
  "run_id": "example-run-id",
  "run_number": "10",
  "run_attempt": "2",
  "runner_environment": "github-hosted"
  "actor": "octocat",
  "workflow": "example-workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch",
  "enterprise": "avocado-corp"
  "ref_type": "branch",
  "job_workflow_ref": "octo-org/octo-
automation/.github/workflows/oidc.yml@refs/heads/main",
  "iss": "https://token.actions.githubusercontent.com",
  "nbf": 1632492967,
  "exp": 1632493867,
  "iat": 1632493567
}
```

To see all the claims supported by GitHub's OIDC provider, review the `claims_supported` entries at https://token.actions.githubusercontent.com/.well-known/openid-configuration.

The token includes the standard audience, issuer, and subject claims.

Claim	Claim type	Description
`aud`	Audience	By default, this is the URL of the repository owner, such as the organization that owns the repository. You can set a custom audience with a toolkit command: `core.getIDToken(audience)`
`iss`	Issuer	The issuer of the OIDC token: `https://token.actions.githubusercontent.com`
`sub`	Subject	Defines the subject claim that is to be validated by the cloud provider. This setting is essential for making sure that access tokens are only allocated in a predictable way.

The OIDC token also includes additional standard JOSE header parameters and claims.

Header Parameter	Parameter type	Description

`alg`	Algorithm	The algorithm used by the OIDC provider.
`kid`	Key identifier	Unique key for the OIDC token.
`typ`	Type	Describes the type of token. This is a JSON Web Token (JWT).

Claim	**Claim type**	**Description**
`exp`	Expires at	Identifies the expiry time of the JWT.
`iat`	Issued at	The time when the JWT was issued.
`jti`	JWT token identifier	Unique identifier for the OIDC token.
`nbf`	Not before	JWT is not valid for use before this time.

The token also includes custom claims provided by GitHub.

Claim	**Description**
`actor`	The personal account that initiated the workflow run.
`actor_id`	The ID of personal account that initiated the workflow run.
`base_ref`	The target branch of the pull request in a workflow run.
`enterprise`	The name of the enterprise that contains the repository from where the workflow is running.
`environment`	The name of the environment used by the job. To include the `environment` claim you must reference an environment.
`event_name`	The name of the event that triggered the workflow run.
`head_ref`	The source branch of the pull request in a workflow run.
`job_workflow_ref`	For jobs using a reusable workflow, the ref path to the reusable workflow. For more information, see "Using OpenID Connect with reusable workflows."
`job_workflow_sha`	For jobs using a reusable workflow, the commit SHA for the reusable workflow file.
`ref`	*(Reference)* The git ref that triggered the workflow run.
`ref_type`	The type of `ref`, for example: "branch".

`repository_visibility`	The visibility of the repository where the workflow is running. Accepts the following values: `internal`, `private`, or `public`.
`repository`	The repository from where the workflow is running.
`repository_id`	The ID of the repository from where the workflow is running.
`repository_owner`	The name of the organization in which the `repository` is stored.
`repository_owner_id`	The ID of the organization in which the `repository` is stored.
`run_id`	The ID of the workflow run that triggered the workflow.
`run_number`	The number of times this workflow has been run.
`run_attempt`	The number of times this workflow run has been retried.
`runner_environment`	The type of runner used by the job. Accepts the following values: `github-hosted` or `self-hosted`.
`workflow`	The name of the workflow.
`workflow_ref`	The ref path to the workflow. For example, `octocat/hello-world/.github/workflows/my-workflow.yml@refs/heads/my_branch`.
`workflow_sha`	The commit SHA for the workflow file.

Defining trust conditions on cloud roles using OIDC claims

With OIDC, a GitHub Actions workflow requires a token in order to access resources in your cloud provider. The workflow requests an access token from your cloud provider, which checks the details presented by the JWT. If the trust configuration in the JWT is a match, your cloud provider responds by issuing a temporary token to the workflow, which can then be used to access resources in your cloud provider. You can configure your cloud provider to only respond to requests that originate from a specific organization's repository. You can also specify additional conditions, described below.

Audience and Subject claims are typically used in combination while setting conditions on the cloud role/resources to scope its access to the GitHub workflows.

- **Audience**: By default, this value uses the URL of the organization or repository owner. This can be used to set a condition that only the workflows in the specific organization can access the cloud role.
- **Subject**: By default, has a predefined format and is a concatenation of some of the key metadata about the workflow, such as the GitHub organization, repository, branch, or associated `job` environment. See "Example subject claims" to see how the subject claim is assembled from concatenated metadata.

If you need more granular trust conditions, you can customize the issuer (`iss`) and subject (`sub`) claims that are included with the JWT. For more information, see "Customizing the token claims".

There are also many additional claims supported in the OIDC token that can be used for setting these conditions. In addition, your cloud provider could allow you to assign a role to the access tokens, letting you specify even more granular permissions.

> **Note**: 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.

Example subject claims

The following examples demonstrate how to use "Subject" as a condition, and explain how the "Subject" is assembled from concatenated metadata. The subject uses information from the `job` context, and instructs your cloud provider that access token requests may only be granted for requests from workflows running in specific branches, environments. The following sections describe some common subjects you can use.

Filtering for a specific environment

The subject claim includes the environment name when the job references an environment.

You can configure a subject that filters for a specific environment name. In this example, the workflow run must have originated from a job that has an environment named `Production`, in a repository named `octo-repo` that is owned by the `octo-org` organization:

- Syntax: `repo:<orgName/repoName>:environment:<environmentName>`
- Example: `repo:octo-org/octo-repo:environment:Production`

Filtering for `pull_request` events

The subject claim includes the `pull_request` string when the workflow is triggered by a pull request event, but only if the job doesn't reference an environment.

You can configure a subject that filters for the `pull_request` event. In this example, the workflow run must have been triggered by a `pull_request` event in a repository named `octo-repo` that is owned by the `octo-org` organization:

- Syntax: `repo:<orgName/repoName>:pull_request`
- Example: `repo:octo-org/octo-repo:pull_request`

Filtering for a specific branch

The subject claim includes the branch name of the workflow, but only if the job doesn't reference an environment, and if the workflow is not triggered by a pull request event.

You can configure a subject that filters for a specific branch name. In this example, the workflow run must have originated from a branch named `demo-branch`, in a repository named `octo-repo` that is owned by the `octo-org` organization:

- Syntax: `repo:<orgName/repoName>:ref:refs/heads/branchName`
- Example: `repo:octo-org/octo-repo:ref:refs/heads/demo-branch`

Filtering for a specific tag

The subject claim includes the tag name of the workflow, but only if the job doesn't reference an environment, and if the workflow is not triggered by a pull request event.

You can create a subject that filters for specific tag. In this example, the workflow run must have originated with a tag named `demo-tag`, in a repository named `octo-repo` that is owned by the `octo-org` organization:

- Syntax: `repo:<orgName/repoName>:ref:refs/tags/<tagName>`
- Example: `repo:octo-org/octo-repo:ref:refs/tags/demo-tag`

Configuring the subject in your cloud provider

To configure the subject in your cloud provider's trust relationship, you must add the subject string to its trust configuration. The following examples demonstrate how various cloud providers can accept the same `repo:octo-org/octo-repo:ref:refs/heads/demo-branch` subject in different ways:

Cloud provider	Example
Amazon Web Services	`"token.actions.githubusercontent.com:sub": "repo:octo-org/octo-repo:ref:refs/heads/demo-branch"`
Azure	`repo:octo-org/octo-repo:ref:refs/heads/demo-branch`
Google Cloud Platform	`(assertion.sub=='repo:octo-org/octo-repo:ref:refs/heads/demo-branch')`
HashiCorp Vault	`bound_subject="repo:octo-org/octo-repo:ref:refs/heads/demo-branch"`

For more information, see the guides listed in "Enabling OpenID Connect for your cloud provider."

Updating your actions for OIDC

To update your custom actions to authenticate using OIDC, you can use `getIDToken()` from the Actions toolkit to request a JWT from GitHub's OIDC provider. For more information, see "OIDC Token" in the npm package documentation.

You could also use a `curl` command to request the JWT, using the following environment variables.

Variable	Description
`ACTIONS_ID_TOKEN_REQUEST_URL`	The URL for GitHub's OIDC provider.
`ACTIONS_ID_TOKEN_REQUEST_TOKEN`	Bearer token for the request to the OIDC provider.

For example:

Shell

```shell
curl -H "Authorization: bearer $ACTIONS_ID_TOKEN_REQUEST_TOKEN"
"$ACTIONS_ID_TOKEN_REQUEST_URL&audience=api://AzureADTokenExchange"
```

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."

Customizing the token claims

You can security harden your OIDC configuration by customizing the claims that are included with the JWT. These customizations allow you to define more granular trust conditions on your cloud roles when allowing your workflows to access resources hosted in the cloud:

- You can customize values for `issuer` or `audience` claims. For more information, see "Customizing the `issuer` value for an enterprise" and "Customizing the `audience` value."
- You can customize the format of your OIDC configuration by setting conditions on the subject (`sub`) claim that require JWT tokens to originate from a specific repository, reusable workflow, or other source.
- You can define granular OIDC policies by using additional OIDC token claims, such as `repository_id` and `repository_visibility`. For more information, see "About security hardening with OpenID Connect".

Customizing the `audience` value

When you use custom actions in your workflows, those actions may use the GitHub Actions Toolkit to enable you to supply a custom value for the `audience` claim. Some cloud providers also use this in their official login actions to enforce a default value for the `audience` claim. For example, the GitHub Action for Azure Login provides a default `aud` value of `api://AzureADTokenExchange`, or it allows you to set a custom `aud` value in your workflows. For more information on the GitHub Actions Toolkit, see the OIDC token section in the documentation.

If you do not want to use the default `aud` value offered by an action, you can provide a custom value for the `audience` claim. This allows you to set a condition that only workflows in a specific repository or organization can access the cloud role. If the action you are using supports this, you can use the `with` keyword in your workflow to pass a custom `aud` value to the action. For more information, see "Metadata syntax for GitHub Actions."

Customizing the `issuer` value for an enterprise

By default, the JWT is issued by GitHub's OIDC provider at `https://token.actions.githubusercontent.com`. This path is presented to your cloud provider using the `iss` value in the JWT.

To security harden their OIDC configuration, enterprise administrators can configure their enterprise to receive tokens from a unique URL at `https://token.actions.githubusercontent.com/<enterpriseSlug>`, replacing `<enterpriseSlug>` with the slug value of the enterprise.

This configuration means that your enterprise will receive the OIDC token from a unique URL, and you can then configure your cloud provider to only accept tokens from that URL. This helps ensure that only the enterprise's repositories can access your cloud resources using OIDC.

To activate this setting for your enterprise, an enterprise administrator must use the `/enterprises/{enterprise}/actions/oidc/customization/issuer` endpoint and specify `"include_enterprise_slug": true` in the request body. For more information, see "GitHub Actions OIDC" in the REST API documentation.

After this setting is applied, the JWT will contain the updated `iss` value. In the following example, the `iss` key uses `octocat-inc` as its `enterpriseSlug` value:

```
{
  "jti": "6f4762ed-0758-4ccb-808d-ee3af5d723a8"
  "sub": "repo:octocat-inc/private-server:ref:refs/heads/main"
  "aud": "http://octocat-inc.example/octocat-inc"
  "enterprise": "octocat-inc"
  "iss": "https://token.actions.githubusercontent.com/octocat-inc",
  "bf": 1755350653,
  "exp": 1755351553,
  "iat": 1755351253
}
```

Customizing the subject claims for an organization or repository

To help improve security, compliance, and standardization, you can customize the standard claims to suit your required access conditions. If your cloud provider supports conditions on subject claims, you can create a condition that checks whether the `sub` value matches the path of the reusable workflow, such as `"job_workflow_ref:octo-org/octo-automation/.github/workflows/oidc.yml@refs/heads/main"`. The exact format will vary depending on your cloud provider's OIDC configuration. To configure the matching condition on GitHub, you can can use the REST API to require that the `sub` claim must always include a specific custom claim, such as `job_workflow_ref`. You can use the OIDC REST API to apply a customization template for the OIDC subject claim; for example, you can require that the `sub` claim within the OIDC token must always include a specific custom claim, such as `job_workflow_ref`.

Note: When the organization template is applied, it will not affect any workflows in existing repositories that already use OIDC. For existing repositories, as well as any new repositories that are created after the template has been applied, the repository owner will need to use the REST API to opt in to receive this configuration. Alternatively, the repository owner could use the REST API to apply a different configuration specific to the repository. For more information, see "GitHub Actions OIDC."

Customizing the claims results in a new format for the entire `sub` claim, which replaces the default predefined `sub` format in the token described in "About security hardening with OpenID Connect."

The following example templates demonstrate various ways to customize the subject claim. To configure these settings on GitHub, admins use the REST API to specify a list of claims that must be included in the subject (`sub`) claim.

To apply this configuration, submit a request to the API endpoint and include the required configuration in the request body. For organizations, see "GitHub Actions OIDC, and for repositories, see "GitHub Actions OIDC."

To customize your subject claims, you should first create a matching condition in your cloud provider's OIDC configuration, before customizing the configuration using the REST API. Once the configuration is completed, each time a new job runs, the OIDC token generated during that job will follow the new customization template. If the matching condition doesn't exist in the cloud provider's OIDC configuration before the job runs, the generated token might not be accepted by the cloud provider, since the cloud conditions may not be synchronized.

Example: Allowing repository based on visibility and owner

This example template allows the `sub` claim to have a new format, using `repository_owner` and `repository_visibility`:

```
{
   "include_claim_keys": [
       "repository_owner",
       "repository_visibility"
   ]
}
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require that claims must include specific values for `repository_owner` and `repository_visibility`. For example: `"sub": "repository_owner:monalisa:repository_visibility:private"`. The approach lets you restrict cloud role access to only private repositories within an organization or enterprise.

Example: Allowing access to all repositories with a specific owner

This example template enables the `sub` claim to have a new format with only the value of `repository_owner`.

To apply this configuration, submit a request to the API endpoint and include the required configuration in the request body. For organizations, see "GitHub Actions OIDC, and for repositories, see "GitHub Actions OIDC."

```
{
   "include_claim_keys": [
       "repository_owner"
```

```
    ]
}
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require that claims must include a specific value for `repository_owner`. For example: `"sub": "repository_owner:monalisa"`

Example: Requiring a reusable workflow

This example template allows the `sub` claim to have a new format that contains the value of the `job_workflow_ref` claim. This enables an enterprise to use reusable workflows to enforce consistent deployments across its organizations and repositories.

To apply this configuration, submit a request to the API endpoint and include the required configuration in the request body. For organizations, see "GitHub Actions OIDC," and for repositories, see "GitHub Actions OIDC."

```
  {
     "include_claim_keys": [
         "job_workflow_ref"
     ]
  }
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require that claims must include a specific value for `job_workflow_ref`. For example: `"sub": "job_workflow_ref:octo-org/octo-automation/.github/workflows/oidc.yml@refs/heads/main"`.

Example: Requiring a reusable workflow and other claims

The following example template combines the requirement of a specific reusable workflow with additional claims.

To apply this configuration, submit a request to the API endpoint and include the required configuration in the request body. For organizations, see "GitHub Actions OIDC," and for repositories, see "GitHub Actions OIDC."

This example also demonstrates how to use `"context"` to define your conditions. This is the part that follows the repository in the default `sub` format. For example, when the job references an environment, the context contains: `environment:<environmentName>`.

```
{
   "include_claim_keys": [
       "repo",
       "context",
       "job_workflow_ref"
   ]
}
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require that claims must include specific values for `repo`, `context`, and `job_workflow_ref`.

This customization template requires that the `sub` uses the following format: `repo:<orgName/repoName>:environment:<environmentName>:job_workflow_ref:<reusableWorkflowPath>`. For example: `"sub": "repo:octo-org/octo-repo:environment:prod:job_workflow_ref:octo-org/octo-automation/.github/workflows/oidc.yml@refs/heads/main"`

Example: Granting access to a specific repository

This example template lets you grant cloud access to all the workflows in a specific repository, across all branches/tags and environments. To further improve security, you can combine this template with a unique issuer URL for your enterprise, as described in "Switching to a unique token URL."

To apply this configuration, submit a request to the API endpoint and include the required configuration in the request body. For organizations, see "GitHub Actions OIDC," and for repositories, see "GitHub Actions OIDC."

```
{
   "include_claim_keys": [
       "repo"
   ]
}
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require a `repo` claim that matches the required value.

Example: Using system-generated GUIDs

This example template enables predictable OIDC claims with system-generated GUIDs that do not change between renames of entities (such as renaming a repository).

To apply this configuration, submit a request to the API endpoint and include the required configuration in the request body. For organizations, see "GitHub Actions OIDC," and for repositories, see "GitHub Actions OIDC."

```
  {
     "include_claim_keys": [
         "repository_id"
     ]
  }
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require a `repository_id` claim that matches the required value.

or:

```
{
   "include_claim_keys": [
       "repository_owner_id"
   ]
}
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require a `repository_owner_id` claim that matches the required value.

Resetting your customizations

This example template resets the subject claims to the default format. This template effectively opts out of any organization-level customization policy.

To apply this configuration, submit a request to the API endpoint and include the required configuration in the request body. For organizations, see "GitHub Actions OIDC," and for repositories, see "GitHub Actions OIDC."

```
{
   "include_claim_keys": [
       "repo",
       "context"
   ]
}
```

In your cloud provider's OIDC configuration, configure the `sub` condition to require that claims must include specific values for `repo` and `context`.

Using the default subject claims

Default subject claims can be created at the organization level. All repositories in an organization have the ability to opt in or opt out of using their organization's default `sub` claim.

To create a default `sub` claim at the organization level, an organization administrator must use the REST API endpoint at "GitHub Actions OIDC." Once an organization has created a default claim, the REST API can be used to programmatically apply the default claim to repositories within the organization. To configure repositories to use the default `sub` claim format, use the REST API endpoint at "GitHub Actions OIDC" with the following request body:

```
{
   "use_default": true
}
```

Example: Configuring a repository to use an organization template

A repository administrator can configure their repository to use the template created by the administrator of their organisation.

To configure the repository to use the organization's template, a repository admin must use the REST API endpoint at "GitHub Actions OIDC" with the following request body:

```
{
   "use_default": false
}
```

Updating your workflows for OIDC

You can now update your YAML workflows to use OIDC access tokens instead of secrets. Popular cloud providers have published their official login actions that make it easy for you to get started with OIDC. For more information about updating your workflows, see the cloud-specific guides listed below in "Enabling OpenID Connect for your cloud provider."

Enabling OpenID Connect for Python package publishing

You can use a GitHub Actions workflow in a repository as a trusted publisher for a PyPI project. Using a workflow as a trusted publisher allows OIDC access tokens to be exchanged for temporary PyPI API tokens. For more information, see "Configuring OpenID Connect in PyPI" and "Publishing to PyPI with a Trusted Publisher" in the PyPI documentation.

Enabling OpenID Connect for your cloud provider

To enable and configure OIDC for your specific cloud provider, see the following guides:

- "Configuring OpenID Connect in Amazon Web Services"
- "Configuring OpenID Connect in Azure"
- "Configuring OpenID Connect in Google Cloud Platform"
- "Configuring OpenID Connect in HashiCorp Vault"

To enable and configure OIDC for another cloud provider, see the following guide:

- "Configuring OpenID Connect in cloud providers"