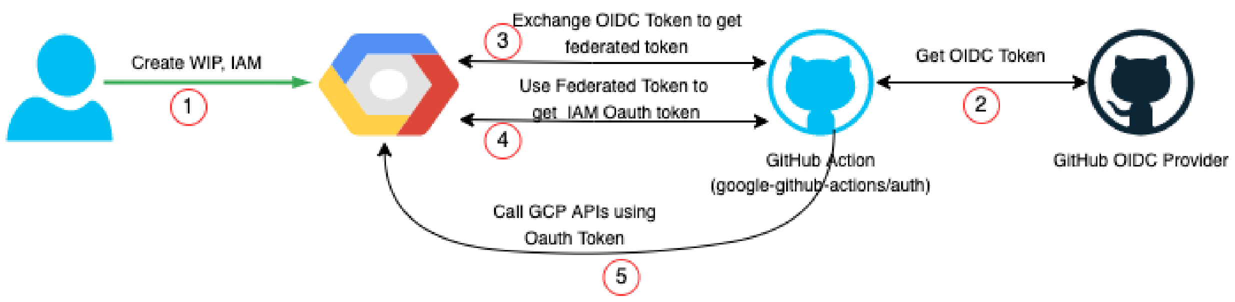
Published in Google Cloud - Community

Pradeep Kumar Singh
Jul 22, 2022 · 5 min read · Listen

How does the GCP Workload Identity Federation work with Github Provider?


Create WIP, IAM
1
3
Exchange OIDC Token to get federated token
Use Federated Token to get IAM Oauth token
4
Get OIDC Token
2
GitHub Action (google-github-actions/auth)
GitHub OIDC Provider
Call GCP APIs using Oauth Token
5

Some users might want to run terraform code in order to create resources on Google cloud Platform through their Github CI/CD pipeline. While others might want to run 'gcloud' commands in order to make certain configurations. Both of these options need to call GCP APIs. And GCP apis need authentication and authorisation when you call them. One possible (but not recommended) way can be to download GCP Service Account Keys and store them in Github repos as secrets or on Github private runners as files. These service account keys then can be used to make GCP apis calls. Securing, storing, distributing, rotating, monitoring these keys in the Github environment can be a very challenging and hectic task in itself. That's why it's not recommended. GCP provides a safer way to achieve the same using Workload Identity Federation. In this article I will try to describe how GCP WIF works with Github Provider using a step wise step approach.

Step 1

In this, the user asks GCP to trust Github Provider. This is done by creating Workload Identity Pool, Workload Identity Provider and IAMs. Let's see each one of these separately.

Workload Identity Pool: Workload identity pools are used to organise and manage external identities. It is recommended to create a new pool for different non google cloud environments. Below command can be used to create the same:

```
gcloud iam workload-identity-pools create github-wif-pool --location="global"
--project <project_id>
```

Workload Identity Provider: Workload Identity Provider describes the relationship between an external identity such as Github and Google Cloud. It basically establishes trust between external identity and GCP. It provides attribute mapping that applies the attributes from an external token to a Google token. This lets IAM use tokens from external providers to authorize access to Google Cloud resources. This is basically a way to translate external tokens into GCP equivalent tokens. Below command can be used to create the same:

```
gcloud iam workload-identity-pools providers create-oidc githubwif \
--location="global" --workload-identity-pool="github-wif-pool"  \
--issuer-uri="https://token.actions.githubusercontent.com" \
--attribute-
mapping="attribute.actor=assertion.actor,google.subject=assertion.sub,attribut
e.repository=assertion.repository" \
--project <project_id>
```

Service Account and IAMs: We need a service account with relevant permissions assigned. WIF will impersonate this service account. We also add permissions to allow authentications from the Workload Identity Provider provided identity to impersonate the desired Service Account. It allows Github action to impersonate the service account and get a token. Please note the use of attribute 'repository' in member name. It allows IAM to authenticate only requests coming from the repository 'PradeepSingh1988/gcp-wif'. If we have not used this, then all the github repos which are using a specific workload identity provider can authenticate against IAM. Below command can be used to create the same:

```
gcloud iam service-accounts create test-wif \
--display-name="Service account used by WIF POC" \
--project <project_id>

gcloud projects add-iam-policy-binding <project_id> \
--member='serviceAccount:test-wif@<project_id>.iam.gserviceaccount.com' \
--role="roles/compute.viewer"

gcloud iam service-accounts add-iam-policy-binding test-
wif@<project_id>.iam.gserviceaccount.com \
--project=<project_id> \
--role="roles/iam.workloadIdentityUser" \
--
member="principalSet://iam.googleapis.com/projects/<project_number>/locations/
global/workloadIdentityPools/github-wif-
pool/attribute.repository/PradeepSingh1988/gcp-wif"
```

There are options to restrict authentication from specific branches too. For that we need to use the principal like below.

```
principal://iam.googleapis.com/projects/<project_number>/locations/global/work
loadIdentityPools/POOL_ID/subject/repo:PradeepSingh1988/gcp-
wif:ref:refs/heads/main
```

This ensures that only the main branch of repo PradeepSingh1988/gcp-wif can authenticate.

Step 2

For this step I would like to give an example of a simple github workflow in my repo which has below steps.

```
name: wif-ci
on:
  push:
    branches:

- 'main'
jobs:
  build:
    name: "Test WIF"
    runs-on: ubuntu-latest
    timeout-minutes: 90
    permissions:
      contents: 'read'
      id-token: 'write'
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - id: auth
        uses: google-github-actions/auth@v0.4.0
        with:
          token_format: "access_token"
          create_credentials_file: true
          activate_credentials_file: true
          workload_identity_provider: ${{
secrets.WORKLOAD_IDENTITY_PROVIDER_ID }}
          service_account: ${{ secrets.SERVICE_ACCOUNT }}
          access_token_lifetime: '100s'
      - name: Set up Cloud SDK
        uses: google-github-actions/setup-gcloud@v0.3.0
      - name: set crdential_file
        run: gcloud auth login --cred-
file=${{steps.auth.outputs.credentials_file_path}}
      - name: Run gcloud
        run: gcloud compute instances list --zones us-east4-c
```

Here we are using some secrets like **WORKLOAD_IDENTITY_PROVIDER_ID** and **SERVICE_ACCOUNT**. We need to create these two secrets in the github repo. We can get their values from step 1.

In this step Github action 'google-github-actions/auth' is first calling Github OIDC provider to get OIDC token. To get an OIDC token it must have permissions to make API calls against the

OIDC provider. That's where the `id-token: **'write'**` line becomes necessary. It provides 'google-github-actions/auth' action necessary permissions to make API calls to OIDC provider.

Github action makes a call to Github OIDC provider. The call is equivalent to the below curl request.

```
OIDC_TOKEN=$(curl -H "Authorization: bearer $ACTIONS_ID_TOKEN_REQUEST_TOKEN"
"$ACTIONS_ID_TOKEN_REQUEST_URL&audience=https://iam.googleapis.com/${{
secrets.WORKLOAD_IDENTITY_PROVIDER_ID }}" | jq '.value')
```

OIDC provider returns token , it is in jwt format and and if decoded successfully, looks something like below:

```
{
"typ": "JWT",
"alg": "RS256",
"x5t": "example-thumbprint",
"kid": "example-key-id"
}

{
"jti": "example-id",
"sub": "repo:PradeepSingh1988/gcp-wif:ref:refs/heads/main"
"environment": "",
"aud":
"https://iam.googleapis.com/projects/<project_number>/locations/global/workloa
dIdentityPools/<pool_id>/providers/<provider_id>",
"ref": "refs/heads/main",
"sha": "example-sha",
"repository": "PradeepSingh1988/gcp-wif",
"repository_owner": "PradeepSingh1988",
"actor_id": "23",
"repository_id": "45",
"repository_owner_id": "67",
"run_id": "example-run-id",
"run_number": "101",
"run_attempt": "21",
"actor": "PradeepSingh1988",
"workflow": "example-workflow",
"head_ref": "",
"base_ref": "",
"event_name": "workflow_dispatch",
"ref_type": "branch",
"job_workflow_ref":"PradeepSingh1988/gcp-
wif/.github/workflows/ci.yml@refs/heads/main",
"iss": "https://token.actions.githubusercontent.com",
"nbf": 1632494000,
"exp": 1632494900,
"iat": 1632494600
}
```

Step 3

In this step Github action 'google-github-actions/auth' exchanges OIDC provided jwt token with GCP security Token Service to get federated access token. The HTTPs call made by Github action is equivalent to the below curl request.

```
STS_RESPONSE=$(curl -0 -X POST https://sts.googleapis.com/v1/token \
 -H 'Accept: application/json' -H 'Content-Type: application/json' -d "$(cat
<<EOF
 {
     "audience": "//iam.googleapis.com/${{
secrets.WORKLOAD_IDENTITY_PROVIDER_ID }}",
     "grantType": "urn:ietf:params:oauth:grant-type:token-exchange",
     "requestedTokenType" : "urn:ietf:params:oauth:token-type:access_token",
     "scope": "https://www.googleapis.com/auth/cloud-platform",
     "subjectTokenType": "urn:ietf:params:oauth:token-type:jwt",
     "subjectToken": $OIDC_TOKEN
 }
 EOF
 )"
 )
 STS_TOKEN=$(jq '.access_token' <<< "$STS_RESPONSE")
```

Here it is passing two important information, audience and subjectToken. After receiving the request STS verifies the token with the help of Workload Identity Provider. Workload Identity Provider does all the condition checks, attribute mapping specified during provider creation. It checks whether the 'iss' field in the token is the same as 'issuer-uri' passed during creating the Workload Identity provider. The 'aud' field in the token is equal to *"https://iam.googleapis.com/projects/<project_number>/locations/global/workloadIdentityPools/<pool_id>/providers/<provider_id>"* or not and many more. All the steps are described here in detail.

Once the request is verified successfully, STS returns a federated token. This token is a kind of GCP identity with all the necessary information necessary for impersonating a service account.

Step 4

In this step Github action 'google-github-actions/auth' exchanges federated token received in previous step to get IAM access token. The HTTPs call made by Github action is equivalent to the below curl request.

```
IAM_RESPONSE=$(curl -0 -X POST   \
https://iamcredentials.googleapis.com/v1/projects/-/serviceAccounts/${{ \
secrets.SERVICE_ACCOUNT }}:generateAccessToken \
-H "Authorization: Bearer $STS_TOKEN" \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' -d "$(cat <<EOF
{
  "scope": [ "https://www.googleapis.com/auth/cloud-platform" ]
}
EOF
)"
)
ACCESS_TOKEN=$(jq '.accessToken' <<< "$IAM_RESPONSE")
```

IAM validates the federated token and checks if it belongs to the correct principal, if yes then it issues an oauth token, which can be used to make GCP api calls.

Step 5

Using the access token received in step 4 workflow makes an API request to GCP for listing instances. This token comes with a short lifespan. Once expired we need to refresh it again.

The process is same for other OIDC Identity Providers as well. I hope you found this article useful. There is a very nice article here as well which puts more details. Please check that out.

Happy Reading!! Comments and suggestions are welcome!!

Google Cloud Platform Workload Identity Google Cloud Security Github Gcp App Dev

2

19 | 2 |

More from Google Cloud - Community

Follow

A collection of technical articles and blogs published or curated by Google Cloud Developer Advocates. The views expressed are those of the authors and don't necessarily reflect those of Google.

Daniel Sanche · Jan 2, 2018

Kubernetes 101: Pods, Nodes, Containers, and Clusters

Sandeep Dinesh · Mar 11, 2018

Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what?

Deepti Garg · Feb 12, 2021

How to work with Arrays and Structs in Google BigQuery

Jitendra Gupta · Jan 26

Google Cloud Landing Zone — Architecture Design

guillaume blaquiere · Jan 18

EventSync: the event-driven management missing piece.

Read more from Google Cloud - Community