ChatGPT

Google Cloud Workload Identity Federation (WIF) 인증 이슈 해결 가이드북

1. 문제 정의

GitHub Actions 기반 CI/CD 파이프라인에서 **Google Cloud Workload Identity Federation (WIF)** 방식으로 Cloud Run에 배포를 시도하는 과정에서 인증 문제가 발생했습니다. 일반적으로 WIF를 사용하면 GitHub Actions 워크플로우가 **서비스 계정 키 없이** OIDC 토큰으로 GCP에 인증할 수 있어 보안에 유리합니다 [1]. 그러나 실제 설정 시 **OIDC 토큰의 대상(audience) 불일치**, **WIF 풀/프로바이더 설정 오류**, **필요 권한 누락** 등으로 인해 배포 단계에서 "권한 없음" 오류(예: 403 Access Denied 또는 401 Unauthenticated)가 발생하였습니다. 이로 인해 CI 파이프라인이 Cloud Run 배포에 실패하는 상황입니다.

이 가이드북은 이러한 WIF 인증 문제를 해결하기 위한 **실무 지침**을 제공합니다. 우선 문제가 발생한 원인을 요약하면 다음과 같습니다:

- **OIDC 토큰 매칭 오류:** GitHub Actions에서 발행한 OIDC 토큰이 GCP Workload Identity Provider가 예상하는 audience 또는 subject와 일치하지 않아 신뢰되지 않았습니다. 예를 들어 **audience 값 불일치** 또는 **subject(claim) 조건 미충족** 문제가 있었습니다.
- **권한 구성 미비:** WIF를 통해 인증된 주체(principal)에 필요한 IAM 권한이 누락되었습니다. 특히 **서비스계정에 대한 impersonation 권한 (roles/iam.workloadIdentityUser)**이나 Cloud Run 배포 권한 (roles/run.admin) 등이 제대로 부여되지 않아 인증 후 실제 GCP API 호출에 실패했습니다.
- **설정 복잡성:** Workload Identity Pool/Provider와 GitHub OIDC 연동 설정이 복잡하여 초기 구성에 실수가 있었습니다 (예: 잘못된 Provider attribute 조건, 잘못된 프로젝트 번호 사용 등).

위 상황으로 **CI/CD 파이프라인이 중단**되었기 때문에, 즉각적인 대응과 근본적인 해결이 모두 필요한 상태입니다.

2. 단기 해결 절차 (임시 대응: 서비스 계정 키 이용)

문제 원인이 WIF 설정 오류로 파악된 경우, **배포 중단을 최소화하기 위해 단기적으로 서비스 계정 키(JSON)**를 사용하는 수동 인증 방식으로 우회할 수 있습니다. 이 방법은 보안상 권장되지는 않지만, WIF 문제가 해결될 때까지 임시로 배포를 진행하는 데 유용합니다 [1]. **절차는 다음과 같습니다:**

1. **서비스 계정 키 발급 및 Secrets 등록:**
2. GCP 콘솔에서 Cloud Run 배포용 서비스 계정의 **키(JSON 파일)**를 새로 발급받습니다. (GCP > IAM > 서비스 계정 > 키 만들기)
3. 발급된 JSON 키 파일의 내용을 GitHub 리포지토리의 Settings > Secrets에 업로드합니다. 예를 들어 `GCP_SA_KEY`라는 이름으로 등록합니다 [2]. 이때 JSON 전체 내용을 시크릿 값으로 넣습니다.
4. 추가로 CI에서 필요한 GCP 프로젝트 ID, Cloud Run 서비스명, 리전 등의 값도 Secrets에 저장합니다 (예: `GCP_PROJECT_ID`, `CLOUD_RUN_SERVICE`, `CLOUD_RUN_REGION` 등) [3].
5. **GitHub Actions 워크플로우 수정:**
6. WIF 인증 부분을 일시적으로 비활성화하거나 대체합니다. 예를 들어 `google-github-actions/auth@v2` 액션을 사용한 부분을 주석 처리합니다.

7. 대신 **서비스 계정 키로 gcloud 인증**을 수행하도록 추가합니다. GitHub에서 제공하는 액션 `google-github-actions/setup-gcloud@v1`을 활용하면 편리합니다.
8. 예시:

```
steps:
 - uses: actions/checkout@v3
 - name: Set up gcloud with SA key
  uses: google-github-actions/setup-gcloud@v1
  with:
   service_account_key: ${{ secrets.GCP_SA_KEY }}
   project_id: ${{ secrets.GCP_PROJECT_ID }}
```

9. 위와 같이 설정하면 해당 러너에 gcloud CLI가 설치되고, 서비스 계정 키를 이용해 GCP 인증이 이뤄집니다 [4] [5]. 이 후 단계에서 `gcloud` 명령이나 `google-github-actions/deploy-cloudrun` 액션을 사용하면 인증된 상태로 Cloud Run에 접근 가능합니다.

10. **배포 실행:**

11. 수정된 워크플로우를 커밋하고 GitHub Actions를 재실행하면, 이번에는 JSON 키를 통해 인증이 진행되어 Cloud Run 배포가 성공할 것입니다.
12. 배포 성공을 확인한 후, **키 파일은 임시 방편**임을 인지하고 유출되지 않도록 주의합니다. 장기적으로 WIF로 전환할 것이므로, 이 키는 **단기간 사용** 후 폐기할 계획을 세웁니다.

    ⚠ **주의:** 서비스 계정 키 방식은 노출 위험과 관리 부담이 있습니다. 키 유출 시 외부에서 GCP 자원에 접근할 수 있으므로, 키를 안전하게 보관하고 필요 최소한의 권한만 가진 서비스 계정을 사용해야 합니다. WIF 문제를 해결한 후에는 해당 JSON 키를 **반드시 폐기**하는 것이 좋습니다.

3. 장기 해결 전략 (WIF 정상화)

근본적인 해결책은 **Workload Identity Federation 설정을 올바르게 구성**하여 키 없는 인증을 안정적으로 적용하는 것입니다. 단계를 요약하면 다음과 같습니다:

3.1 Workload Identity Pool 및 Provider 재구성

1. **Workload Identity Pool 생성:**
   GCP IAM에서 GitHub Actions용 **Workload Identity Pool**을 생성합니다. Pool은 WIF의 기본 컨테이너로, 전역(global) 위치에 하나 만들어 두는 것이 일반적입니다.
2. CLI 예시:

```
gcloud iam workload-identity-pools create "github-pool" \
 --project "<GCP_PROJECT_ID>" \
 --location "global" \
 --display-name "GitHub Actions Pool"
```

3. `github-pool`이라는 이름의 풀을 생성하고, 프로젝트 ID와 위치 등을 지정합니다. (GCP 콘솔 GUI로도 생성 가능함)

4. **OIDC Provider 설정:**
   앞서 만든 Pool 안에 **Workload Identity Provider**를 생성합니다. Provider는 GitHub의 OIDC 토큰을 검증하고 필요한 클레임을 매핑하는 역할을 합니다.

5. 기본 설정 요소:
   - **issuer URI:** GitHub Actions용 OIDC 발급자의 주소는 `https://token.actions.githubusercontent.com` 입니다.
   - **attribute mapping:** GitHub 토큰의 클레임(assertion)을 GCP에서 사용할 속성으로 매핑해야 합니다. 일반적으로 `google.subject=assertion.sub`로 설정하고, 추가로 `attribute.repository=assertion.repository`, `attribute.actor=assertion.actor` 등 필요한 값을 매핑할 수 있습니다.
   - **attribute condition:** (선택) 특정 리포지토리나 브랜치만 신뢰할 경우 조건을 걸 수 있습니다. 예를 들어 `attribute.repository=="<OWNER>/<REPO>"` 조건을 주어 해당 저장소 토큰만 허용하거나, `assertion.sub.endsWith('ref:refs/heads/main')` 조건으로 메인 브랜치 워크플로우만 허용할 수 있습니다.

6. CLI 예시 (특정 저장소로 제한하는 경우):

```
gcloud iam workload-identity-pools providers create-oidc "gha-actions" \
 --project "<GCP_PROJECT_ID>" \
 --location "global" \
 --workload-identity-pool "github-pool" \
 --display-name "GitHub OIDC Provider" \
 --issuer-uri "https://token.actions.githubusercontent.com" \
 --attribute-mapping
"google.subject=assertion.sub,attribute.repository=assertion.repository" \
 --attribute-condition "attribute.repository=='<GitHub_ORG>/<REPO_NAME>'"
```

   위 명령으로 발행자(issuer), 매핑, 조건이 설정된 Provider가 생성됩니다. 여기서는 GitHub `<GitHub_ORG>/<REPO_NAME>`에서 오는 토큰만 신뢰하도록 제한했습니다. (조직 전체를 허용하려면 repository_owner나 여러 저장소 패턴으로 조건을 주거나, 조건을 생략할 수도 있습니다.)

7. Provider 생성 후 **리소스 이름**을 확인합니다. `gcloud iam workload-identity-pools providers describe ... --format="value(name)"` 명령으로 얻을 수 있는데, 보통 형식은:

```
projects/<PROJECT_NUMBER>/locations/global/workloadIdentityPools/github-pool/providers/gha-actions
```

   이 리소스 이름이 GitHub Actions 워크플로우에서 사용됩니다.

   참고: Provider 설정 시 **audience 문제**를 유의해야 합니다. 기본적으로 Google WIF는 GitHub OIDC 토큰의 `aud` 클레임이 Provider 리소스 이름으로 되어야 인증이 원활합니다. `google-github-actions/auth` 액션을 사용하면 내부적으로 이 요구사항에 맞게 `audience`를 설정합니다. 만약 수동으로 OIDC 토큰을 얻는다면 `audience` 파라미터에 위의 Provider 리소스 이름을 지정해야 합니다. 이 부분이 맞지 않으면 “invalid audience” 오류로 인증이 실패할 수 있습니다.

3.2 서비스 계정 구성 및 권한 부여

1. **배포용 서비스 계정 확인/생성:**
   Cloud Run 배포에 사용할 GCP 서비스 계정을 준비합니다. 이미 CI/CD용 서비스 계정 (예: `gha-deploy-sa`)가 있다면 그대로 사용하고, 없다면 새로 만듭니다.

2. 해당 서비스 계정에 Cloud Run 배포에 필요한 IAM 역할을 부여합니다:

   - Cloud Run Admin 또는 Developer 권한 (`roles/run.admin` 혹은 최소 `roles/run.developer` + `roles/run.serviceAgent` 등) – **Cloud Run 서비스 생성/업데이트** 권한용.
   - Artifact Registry Writer (`roles/artifactregistry.writer`) – **컨테이너 이미지 푸시** 권한용 (이미지 빌드&푸시 시 필요).
   - (필요 시) Cloud Build 권한 (`roles/cloudbuild.builds.editor`) – 만약 gcloud로 빌드/배포 한다면 Cloud Build 사용을 위해 필요할 수 있습니다.
   - 기타 필요한 권한: 프로젝트 Viewer, Cloud Run Invoker (없어도 배포에는 무방) 등 작업에 따라 추가.
   - **서비스 계정 자체에 대한 impersonation 권한**은 다음 단계에서 별도로 설정합니다.

3. **Workload Identity Pool과 서비스 계정 연결 (권한 트러스트 구성):**
   WIF를 통해 GitHub OIDC 주체가 해당 서비스 계정을 **가장(impersonate)**할 수 있도록 IAM 정책 바인딩을 설정합니다. 이는 서비스 계정에 **roles/iam.workloadIdentityUser** 역할을 부여하고, 그 멤버로 Workload Identity Pool의 주체를 지정하는 형태로 이뤄집니다 [6].

4. CLI 예시:

```
gcloud iam service-accounts add-iam-policy-binding "gha-deploy-sa@<PROJECT_ID>.iam.gserviceaccount.com" \
 --role "roles/iam.workloadIdentityUser" \
 --member "principalSet://iam.googleapis.com/projects/<PROJECT_NUMBER>/locations/global/workloadIdentityPools/github-pool/attribute.repository/<GitHub_ORG>/<REPO_NAME>"
```

   위에서 `<PROJECT_NUMBER>`는 GCP 프로젝트의 **숫자 ID**입니다 (프로젝트 이름이 아님에 유의). principalSet 멤버 문자열은 앞서 만든 Pool 내 **특정 조건에 매칭되는 주체들**을 뜻합니다. 여기서는 `attribute.repository/<ORG>/<REPO>`로 지정하여 해당 GitHub 저장소에서 오는 OIDC 토큰에만 이 역할을 부여합니다. (만약 Provider에서 조건을 이미 지정했다면 principalSet 멤버는 조건 없이 풀 전체로 줄 수도 있지만, 보안을 위해 이중으로 특정하는 것이 좋습니다.)

5. 이 설정으로 **GitHub Actions의 OIDC 주체**는 roles/iam.workloadIdentityUser 권한을 통해 `gha-deploy-sa` 서비스계정으로 **인증 토큰 교환**을 할 수 있게 됩니다. 쉽게 말해 "GitHub 워크플로우가 이 서비스계정으로 로그인할 수 있는" 권한을 준 것입니다.

6. **구성 검증:**

7. GCP IAM에서 해당 서비스 계정의 정책을 열어 **Workload Identity User** 역할 항목이 추가되었는지 확인합니다.
8. principalSet 멤버 문자열이 올바르게 표기되었는지 (특히 프로젝트 번호, 풀 이름, 속성 등이 정확한지) 검토합니다.
9. 또한 Workload Identity Provider의 attribute-condition도 우리의 서비스 계정 바인딩 조건과 논리적으로 일치해야 합니다. (예: Provider가 이미 repository 한정이면 principalSet도 repository 한정으로 줌)

여기까지 설정하면 **WIF 구성 완료**입니다. 요약하면, GitHub Actions -> OIDC 토큰 -> Workload Identity Provider (GitHub 신뢰) -> Workload Identity Pool -> 서비스계정 권한 획득의 흐름이 마련되었습니다.

4. GitHub Actions에서의 OIDC 구성과 인증 로그 확인법

이제 GitHub Actions 워크플로우에서 WIF를 사용하도록 설정하고, 올바로 동작하는지 확인하는 방법입니다.

1. **GitHub Actions 워크플로우 YAML 설정:**
2. **permissions 추가:** 해당 워크플로우에 `id-token: write` 권한을 부여해야 GitHub가 OIDC 토큰을 생성합니다 [7]. 예를 들어:

```
permissions:
  contents: read
  id-token: write
```

`contents: read`는 기본 값이고, 반드시 `id-token: write`가 포함되어야 합니다 [8]. 이 설정이 없으면 OIDC 토큰을 받을 수 없습니다.

3. **OIDC 인증 액션 사용:** Google에서 제공하는 `google-github-actions/auth@v2` 액션을 steps에 추가합니다. 이 액션이 GitHub의 OIDC 토큰을 자동으로 교환하여 GCP 액세스 토큰을 획득해 줍니다. 설정 예시:

```
- name: Authenticate to GCP via WIF
  uses: google-github-actions/auth@v2
  with:
    project_id: ${{ secrets.GCP_PROJECT_ID }}
    workload_identity_provider: "projects/<PROJECT_NUMBER>/locations/global/
workloadIdentityPools/github-pool/providers/gha-actions"
    service_account: "gha-deploy-sa@<PROJECT_ID>.iam.gserviceaccount.com"
    token_format: "access_token"
```

위와 같이 `workload_identity_provider`에 앞서 확인한 Provider **리소스 전체 경로**를 입력하고, `service_account`에 가장할 서비스 계정의 이메일을 넣습니다 [9]. `project_id`도 지정해 주면 gcloud 사용 시 기본 프로젝트로 설정되어 편리합니다.
`token_format: "access_token"` 옵션은 GCP API 호출에 사용할 **액세스 토큰**을 얻겠다는 의미입니다. (기본값이며, 명시적으로 적지 않아도 무방합니다. 만약 Cloud Run invoke에 **ID 토큰**이 필요하면 `id_token`으로 요청할 수 있습니다.)

4. **gcloud 세션 설정 (선택):** 이후 GCP CLI나 deploy 액션을 쓸 경우를 대비해 `google-github-actions/setup-gcloud@v2` 액션을 사용해 Cloud SDK를 설치하고 프로젝트를 설정할 수 있습니다. 인증은 이미 위 auth 액션으로 이루어졌으므로 `setup-gcloud`에서는 별도 키가 필요 없습니다:

```
- uses: google-github-actions/setup-gcloud@v2
  with:
    project_id: ${{ secrets.GCP_PROJECT_ID }}
    install_components: 'beta'  # (예: 필요한 gcloud 컴포넌트 설치)
```

이로써 이후 `gcloud` 명령을 자유롭게 사용할 수 있습니다.

5. **Cloud Run 배포 실행:** WIF로 인증된 상태에서 Cloud Run 배포를 수행합니다. 방법은 두 가지입니다:

    - **gcloud CLI 사용:** 예를 들어 `gcloud run deploy` 명령을 실행하는 step을 추가하여 배포합니다. 이때 이미지 경로, 서비스명, 리전, 환경변수 등을 지정해야 합니다.
    - **Deploy Cloud Run 액션 사용:** `google-github-actions/deploy-cloudrun@v0` 액션에 인자를 넣어 배포할 수도 있습니다. (이 액션 내부적으로 build->deploy를 수행합니다.)
    - 어찌되었든, **인증 토큰은 앞 단계에서 확보**되었으므로 배포 단계에서는 권한 문제가 없어야 합니다.

6. **OIDC 토큰 및 인증 확인 (로그 모니터링):**
   WIF 인증이 잘 이루어졌는지 확인하기 위해 몇 가지 방법을 활용합니다:

7. **GitHub Actions 로그 확인:** `google-github-actions/auth@v2` 스텝의 로그를 보면 `"Generated credentials"` 또는 `"Authentication successful"` 등의 메시지가 나타납니다. 에러가 있다면 해당 로그에 자세한 원인이 출력되므로 면밀히 살펴보세요.
8. **디버그 정보 출력:** WIF 설정 문제를 해결하는 과정에서, 토큰의 클레임이나 audience 등을 확인하려면 GitHub Actions에서 OIDC 토큰을 수동으로 가져와 출력해볼 수 있습니다. 예를 들어 다음 명령으로 현재 사용 중인 인증 토큰의 audience를 확인할 수 있습니다 (위 auth 액션이 성공한 경우):

    ```
    - name: Debug OIDC Token
      run: |
        echo "Audience in credentials:"
        jq -r '.credential_source.oidc_token.audience' "$GOOGLE_GHA_CREDS_PATH" || echo "No audience info"
    ```

    `GOOGLE_GHA_CREDS_PATH`는 auth 액션이 생성한 임시 자격증명 JSON 경로입니다 [10]. 여기서 추출한 audience가 우리가 의도한 Provider 리소스와 일치하는지 확인합니다. 또한 `assertion.repository` 등의 클레임도 `jq`로 확인 가능하니, 필요하다면 전체 `$ACTIONS_ID_TOKEN` (GitHub가 발급한 JWT)을 디코드해서 검증해 봅니다.
9. **GCP Cloud Logging 확인:** Workload Identity Federation을 통한 STS(Security Token Service) 교환은 GCP의 **Audit Log**에 남습니다. GCP Cloud Console의 로그에서 `principalSubject`나 `AuthenticateToken` 등의 항목을 필터링하면, GitHub OIDC 토큰이 교환된 내역과 오류를 확인할 수 있습니다. 예를 들어 “Invalid audience” 등의 메시지가 audit log에 기록되므로, 이를 통해 원인을 파악할 수 있습니다.
10. **gcloud 인증 상태 검사:** 워크플로우 내에서 `gcloud auth list` 명령을 실행해 현재 인증된 계정을 확인할 수 있습니다. 예시:

    ```
    - run: |
        echo "Active account:"
        gcloud auth list --filter=status:ACTIVE --format="value(account)"
    ```

    출력이 `gha-deploy-sa@...` 서비스계정으로 나오면 WIF 인증이 제대로 적용된 것입니다. 또한 `gcloud auth print-access-token`으로 액세스 토큰이 정상 발급되었는지도 확인할 수 있습니다 [11].

5. Cloud Run 배포 전/후 인증 테스트 절차

WIF 설정 이후 실제 배포가 잘 되는지 사전에 검증하고, 배포 후에는 애플리케이션 접근에 문제가 없는지 테스트하는 것이 좋습니다. 다음과 같은 **전후 테스트 절차**를 권장합니다:

- **사전 테스트 (로컬 또는 CI에서 시뮬레이션):**
  WIF 설정을 마쳤다면 곧바로 배포 워크플로우를 돌리기 전에, 가벼운 **인증 테스트용 워크플로우**를 한 번 실행해 보세요. 예를 들어 manual dispatch가 가능한 워크플로우(`workflow_dispatch`)로 `google-github-actions/auth@v2`만 수행하고 `gcloud projects list`나 `gcloud run services list` 같은 읽기 요청을 해보는 것입니다.
  이 테스트 워크플로우에서 **Authentication succeeded** 메시지와 함께 GCP 리소스를 나열할 수 있다면, WIF 인증은 정상으로 간주할 수 있습니다. 만약 여기서도 실패하면 앞서 구성한 Pool/Provider/권한을 재점검해야 합니다.

- **Cloud Run 배포 테스트:**
  WIF 인증이 성공한 것을 확인했다면, **Cloud Run 배포 단계까지 포함**한 CI 파이프라인을 동작시켜 봅니다. (예: main 브랜치에 dummy 커밋을 푸시하여 deploy 워크플로우 트리거)
  배포 로그에 권한 오류 없이 Cloud Run에 새로운 리비전이 생성되면 CI 관점의 인증 문제는 해결된 것입니다. Cloud Run 배포 시 출력된 서비스 URL을 통해 새 버전의 서비스가 잘 작동하는지도 확인합니다.

- **배포 후 애플리케이션 접근 확인:**
  Cloud Run 서비스를 **public으로 열어둔 경우**(`--allow-unauthenticated`)에는 별도 IAM 인증 없이 접근이 가능하므로 애플리케이션 레벨에서의 토큰 인증만 확인하면 됩니다. 만약 Cloud Run을 **private**으로 설정하여 **IAM 인증이 필요**한 구조라면, 서비스 계정이 올바르게 토큰을 받아 호출할 수 있는지 테스트해야 합니다.

- GCP 콘솔 또는 `gcloud run services invoke` 명령으로 해당 서비스에 요청해보세요.
  `gcloud run services invoke <SERVICE> --project <PROJECT_ID> --region <REGION> --format=json` 등을 사용하면 지정한 서비스계정 자격으로 호출할 수 있습니다.

- 혹은 OAuth 2.0 **ID Token**을 발급받아 직접 `curl`로 호출해 볼 수도 있습니다:

```
gcloud auth print-identity-token --audiences="<CloudRun_URL>" --
impersonate-service-account="gha-deploy-
sa@<PROJECT_ID>.iam.gserviceaccount.com"
curl -H "Authorization: Bearer $(gcloud auth print-identity-token --
audiences="<CloudRun_URL>" --impersonate-service-account="gha-deploy-
sa@...")" <CloudRun_URL>
```

  위 명령으로 해당 서비스계정으로 서명된 ID 토큰을 생성하여 Cloud Run 엔드포인트에 요청을 보내 볼 수 있습니다. 응답이 200으로 오면 인증이 제대로 된 것입니다. (Cloud Run이 public이면 401 대신 애플리케이션 응답이 오겠죠. private이면 올바른 토큰이 없으면 401이 뜰 것입니다.)

- **환경변수/권한 등 부가 점검:**
  마지막으로 Cloud Run에 배포된 서비스의 **환경 변수**(시크릿) 설정이 잘 적용되었는지, 필요한 **서비스 계정 권한(예: 다른 API 접근)**이 누락되지 않았는지도 확인합니다. WIF 자체는 배포 권한만 좌우하지만, 어플리케이션 실행 시 추가 GCP 자원 접근이 필요하다면 해당 서비스 계정(`gha-deploy-sa`)에 적절한 권한이 있는지 점검하세요.

以上의 테스트를 통해 WIF 기반 인증 및 Cloud Run 배포 파이프라인이 정상 동작함을 검증할 수 있습니다.

6. 실패 상황별 Plan B 및 디버깅 체크리스트

WIF 적용 후에도 인증 문제가 발생할 경우, 아래와 같은 **대응책(Plan B)**과 **디버깅 포인트**를 활용하세요:

- **Plan B: JSON 키 재활용** – 여전히 WIF 오류로 배포가 막힐 경우, 2번 절차의 **JSON 키 방식을 임시로 재활용**할 수 있습니다. 즉시 배포가 필요하면 JSON 시크릿을 활성화하고 WIF 문제를 해결한 후 다시 WIF로 전환합니다. 이때 키를 장기간 노출시키지 않도록 주의해야 합니다.

- **디버깅 체크리스트:**

- **GitHub Actions 설정 확인:** 워크플로우 YAML에 `permissions: id-token: write`가 빠지지 않았는지, `auth@v2` 액션에 올바른 provider 경로와 서비스계정이 설정되었는지 확인합니다. 특히 provider 경로는 **프로젝트 번호**를 포함해야 함에 유의합니다 (프로젝트 ID 문자열이 아닙니다).
- **WIF Pool/Provider 설정 재검증:** GCP 콘솔에서 Workload Identity Pool과 Provider의 설정을 다시 봅니다.
    - Issuer URL이 정확히 `https://token.actions.githubusercontent.com`인가?
    - Attribute Mapping에 `google.subject` 등이 올바르게 지정되었는가?
    - Attribute Condition이 있다면 현재 워크플로우의 토큰 클레임과 부합하는가? (조건을 너무 엄격하게 주지는 않았는지 확인; 예를 들어 브랜치 이름이나 워크플로우 이름이 정확히 일치해야 통과하도록 해두면 다른 이벤트에는 실패할 수 있음)
- **IAM 권한 확인:** 서비스 계정에 Cloud Run, Artifact Registry 등의 **Role**이 부여되었는지 확인합니다 (누락 시 해당 서비스 API 호출이 거부됨). 또한 **roles/iam.workloadIdentityUser**가 부여되었는지도 재확인합니다 [6]. 만약 여러 서비스 계정을 사용하는 경우 (예: 빌드용과 배포용 분리) 각각 필요한 권한과 WIF 바인딩이 되어야 합니다.
- **Audit Logs 확인:** GCP의 IAM 또는 STS 관련 에러 로그를 검사합니다. 로그에는 실패 원인이 비교적 명확히 나오므로, 예컨대 "subject 'repo:owner/repo:ref:...:<something>' does not match condition" 이라든가 "audience not allowed" 같은 메시지를 찾습니다. 이러한 정보를 바탕으로 Provider의 조건을 수정하거나, GitHub 쪽 토큰 발행 방법을 조정합니다.
- **OIDC 토큰 수동 테스트:** 로컬에서 `curl` 등을 이용해 GitHub OIDC 토큰을 직접 받아 디코딩해보는 방법도 있습니다. GitHub CLI를 사용하면 `gh auth login` 후 `gh oidc-token <aud>` 명령으로 토큰을 얻을 수 있습니다. 이를 디코드해서 **issuer, subject, audience, expiration** 등을 검토하고 GCP 설정과 맞지 않는 부분을 찾습니다.
- **지원 대상 확인:** Workload Identity Federation이 **모든 Google Cloud API를 지원하는 것은 아닙니다** [12]. 대부분 서비스는 문제 없으나, 만약 특이한 GCP 서비스에 접근하려는 경우 federated token을 직접 받아주지 않는 케이스가 있습니다. Cloud Run 배포, Artifact Registry 등은 WIF 지원이 되므로 큰 문제는 없지만, 만일의 호환성 이슈도 고려하세요. (일부 오래된 gcloud 구성에서는 WIF 토큰으로 `gsutil` 사용이 안 되는 이슈 등이 알려져 있습니다 [13] [14].)

- **기타 일반적인 원인:** 시간 동기화 문제(토큰 만료), GitHub Actions 캐시로 인한 이전 자격증명 잔존 등의 가능성도 있습니다. 항상 fresh run으로 테스트하고, 토큰의 유효기간(5분)을 넘겨 사용하려 하지 않는지 유의하세요 [15].

- **전환 전략:** JSON 키에서 WIF로 전환 시 **점진적**으로 진행하세요. 예를 들어, 테스트 겸해 **정적 사이트**나 **개발 환경** 배포에 먼저 WIF를 적용해 본 후, 안정되면 프로덕션 배포에 확대 적용하는 것이 안전합니다. 문제 발생 시 빠르게 키 방식으로 롤백할 수 있도록 워크플로우에 조건분기를 두는 것도 고려할 만합니다 (예: 입력 매개변수에 따라 auth 방식을 선택).

- **지원 채널 활용:** Google Cloud Support나 커뮤니티를 통해 비슷한 사례의 해결법을 찾아보는 것도 좋습니다. `github-actions workload-identity-federation 403 error` 등의 키워드로 검색하면 유사 이슈와 해결법이 공유되어 있습니다. 공식 문서의 [Troubleshoot Workload Identity Federation 가이드]도 참고하세요.

7. 참고 스크립트, 명령어, GitHub Secrets 구성 예시

아래는 위에서 설명한 절차들을 실행하는 데 도움이 되는 예시들과 참고 자료입니다:

- **Terraform 예시 (선택):** Infrastructure as Code를 선호한다면, Terraform으로 WIF를 설정할 수 있습니다. 예를 들어:

```
resource "google_iam_workload_identity_pool" "github_pool" {
 project      = "<PROJECT_ID>"
 location     = "global"
 display_name  = "GitHub Actions Pool"
 workload_identity_pool_id = "github-pool"
}
resource "google_iam_workload_identity_pool_provider" "github_provider" {
 workload_identity_pool_id =
google_iam_workload_identity_pool.github_pool.workload_identity_pool_id
 location     = "global"
 workload_identity_pool_provider_id = "gha-actions"
 display_name  = "GitHub OIDC Provider"
 issuer_uri    = "https://token.actions.githubusercontent.com"
 attribute_mapping = {
  "google.subject"   = "assertion.sub"
  "attribute.repository" = "assertion.repository"
 }
 attribute_condition = "attribute.repository=='<ORG>/<REPO>'"
}
resource "google_service_account" "deploy_sa" {
 account_id  = "gha-deploy-sa"
 display_name = "GitHub Actions Deploy SA"
}
resource "google_service_account_iam_binding" "wif_bind" {
 service_account_id = google_service_account.deploy_sa.name
 role  = "roles/iam.workloadIdentityUser"
 members = [
  "principalSet://iam.googleapis.com/$
{google_iam_workload_identity_pool.github_pool.name}/attribute.repository/<ORG>/
<REPO>"
 ]
}
```

위 Terraform 코드는 수작업으로 진행한 Pool, Provider, SA 생성과 바인딩을 자동화한 것입니다. Terraform 적용 후 GitHub Actions쪽 설정만 맞추면 WIF를 사용할 수 있게 됩니다.

- **CLI 명령 요약:** (bash 스크립트 형식으로)

```
# 환경 변수 준비
GCP_PROJECT_ID="<프로젝트 ID>"; PROJECT_NUM=$(gcloud projects describe
$GCP_PROJECT_ID --format="value(projectNumber)")
GH_REPO="your-org/your-repo"
SA_NAME="gha-deploy-sa"

# 1. Workload Identity Pool 생성
gcloud iam workload-identity-pools create "github-pool" \
 --project "$GCP_PROJECT_ID" --location="global" \
 --display-name "GitHub Actions Pool"
# 2. Provider 생성
gcloud iam workload-identity-pools providers create-oidc "gha-actions" \
 --project "$GCP_PROJECT_ID" --location="global" \
 --workload-identity-pool "github-pool" \
 --display-name "GitHub OIDC Provider" \
 --issuer-uri "https://token.actions.githubusercontent.com" \
 --attribute-mapping
"google.subject=assertion.sub,attribute.repository=assertion.repository" \
 --attribute-condition "attribute.repository=='${GH_REPO}'"
# Provider 리소스 ID 확인
PROVIDER_RESOURCE=$(gcloud iam workload-identity-pools providers describe
"gha-actions" \
 --project "$GCP_PROJECT_ID" --location="global" --workload-identity-pool
"github-pool" \
 --format="value(name)")
echo "Provider Resource: $PROVIDER_RESOURCE"

# 3. 서비스 계정 생성 (이미 있다면 생략)
gcloud iam service-accounts create "$SA_NAME" \
 --project "$GCP_PROJECT_ID" --display-name "GitHub Actions Deploy SA"
SA_EMAIL="$SA_NAME@$GCP_PROJECT_ID.iam.gserviceaccount.com"
# 필요한 IAM 역할 부여
gcloud projects add-iam-policy-binding "$GCP_PROJECT_ID" \
 --member "serviceAccount:$SA_EMAIL" --role "roles/run.admin"
gcloud projects add-iam-policy-binding "$GCP_PROJECT_ID" \
 --member "serviceAccount:$SA_EMAIL" --role "roles/artifactregistry.writer"
# (필요 시) Cloud Build 권한 등 추가

# 4. WIF Pool -> SA 연결 (IAM binding)
gcloud iam service-accounts add-iam-policy-binding "$SA_EMAIL" \
 --role "roles/iam.workloadIdentityUser" \
 --member "principalSet://iam.googleapis.com/projects/$PROJECT_NUM/locations/global/
workloadIdentityPools/github-pool/attribute.repository/${GH_REPO}"
echo "WIF binding completed for $SA_EMAIL"
```

위 스크립트를 실행하면 WIF 설정과 IAM 권한 부여가 일괄적으로 이루어집니다. 각 단계의 출력과 에러를 확인하면서 진행하세요.

- **GitHub Secrets 구성:**
  WIF 방식에서는 장기 키를 저장할 필요가 없지만, 여전히 몇 가지 값을 Secrets로 관리해야 합니다. 예를 들면:

- `GCP_PROJECT_ID`: GCP 프로젝트 ID (워크플로우에서 project_id 입력 등에 사용)
- `CLOUD_RUN_SERVICE`: 배포 대상 Cloud Run 서비스 이름
- `CLOUD_RUN_REGION`: Cloud Run 리전 (예: asia-northeast3)
- (Artifact Registry 정보를 별도로 쓴다면) `GCP_ARTIFACT_REGISTRY_HOST`: Artifact Registry 도메인 (예: `asia-northeast3-docker.pkg.dev`)

- 애플리케이션 환경변수에 해당하는 시크릿들 (예: `ADMIN_TOKEN`, `API_KEY` 등)
  JSON 키 방식에 썼던 `GCP_SA_KEY`는 WIF 전환 후엔 더 이상 필요치 않으므로 삭제하거나 비활성화하도록 합니다.

- **예시 워크플로우 (발췌):**

```
name: CD to Cloud Run
on:
 push:
  branches: [ "main" ]

permissions:
 contents: read
 id-token: write

jobs:
 build-deploy:
  runs-on: ubuntu-latest
  steps:
   - uses: actions/checkout@v4

   - name: Authenticate to Google Cloud via WIF
     uses: google-github-actions/auth@v2
     with:
      project_id: ${{ secrets.GCP_PROJECT_ID }}
      workload_identity_provider: ${{ secrets.GCP_WIF_PROVIDER }}  # (선택: provider 리소스 경로를 시크릿에 저장한 경우)
      service_account: "gha-deploy-sa@${{ secrets.GCP_PROJECT_ID }}.iam.gserviceaccount.com"

   - name: Setup gcloud
     uses: google-github-actions/setup-gcloud@v2
     with:
      project_id: ${{ secrets.GCP_PROJECT_ID }}
      install_components: beta

   - name: Build & Push Image
     run: |
      gcloud auth configure-docker ${GCP_ARTIFACT_REGISTRY_HOST} -q
      docker build -t ${GCP_ARTIFACT_REGISTRY_HOST}/${{ secrets.GCP_PROJECT_ID }}/${{ secrets.CLOUD_RUN_SERVICE }}:${{ github.sha }} .
```

```
        docker push ${GCP_ARTIFACT_REGISTRY_HOST}/${{ secrets.GCP_PROJECT_ID }}/$
{{ secrets.CLOUD_RUN_SERVICE }}:${{ github.sha }}

    - name: Deploy to Cloud Run
      run: |
        gcloud run deploy ${{ secrets.CLOUD_RUN_SERVICE }} \
          --image ${GCP_ARTIFACT_REGISTRY_HOST}/${{ secrets.GCP_PROJECT_ID }}/$
{{ secrets.CLOUD_RUN_SERVICE }}:${{ github.sha }} \
          --platform managed --region ${{ secrets.CLOUD_RUN_REGION }} \
          --allow-unauthenticated \
          --update-env-vars ADMIN_TOKEN=${{ secrets.ADMIN_TOKEN }},API_KEY=$
{{ secrets.API_KEY }}
```

위 예시는 WIF로 GCP에 인증한 뒤 Cloud Run에 컨테이너 이미지를 빌드/푸시하고 배포하는 전체 흐름을 보여줍니다. 각 환경에 맞게 프로젝트, 리전, 서비스이름 등을 조정해야 합니다.

- **추가 참고 자료:**

- Google Cloud 공식 블로그: “Deploy to Cloud Run with GitHub Actions” (OIDC WIF 설정 과정 상세 설명) [16] [17]
- GitHub Actions Auth 액션 문서: google-github-actions/auth README (WIF 설정 방법과 Troubleshooting 팁) [18] [7]
- Google Cloud Docs: Troubleshoot Workload Identity Federation (WIF 오류 메시지별 해결 가이드) [19] [20]

以上의 가이드에 따라 단계별로 수행하면, **GitHub Actions -> Workload Identity Federation -> Cloud Run**으로 이어지는 인증 흐름을 안전하게 구축하고 문제를 해결할 수 있을 것입니다. 설정 완료 후에도 주기적으로 권한을 검토하고, 토큰 사용 로그를 모니터링하여 안정적으로 운영하세요. 감사합니다.

---

[1] [2] [3] [4] [5] [16] [17] 1012_2QualiJournal 관리자 시스템 작업 가이드북.pdf
file://file-Ad1vex4HDdK79hKp9LXwzj

[6] SETUP_WIF.md
https://github.com/julhaas91/boilerplate-cloud-run-python/blob/5b6fdb3d1d10f1e0f5ae8c29ee260b26e46bc4a7/SETUP_WIF.md

[7] [13] [14] [15] [18] GitHub - google-github-actions/auth: A GitHub Action for authenticating to Google Cloud.
https://github.com/google-github-actions/auth

[8] [9] [10] [11] wif-auth-check.yml
https://github.com/cdmacs1003-cyber/quali-journal/blob/46a466572a79f28b58b66cda7d5b7a6273d39b94/.github/workflows/wif-auth-check.yml

[12] [19] [20] Troubleshoot Workload Identity Federation | IAM Documentation | Google Cloud
https://cloud.google.com/iam/docs/troubleshooting-workload-identity-federation