[How to access Files in Google Cloud Storage through GKE pods](https://stackoverflow.com/questions/63664800/how-to-access-files-in-google-cloud-storage-through-gke-pods)

3

I'm trying get image files of Google Cloud Storage (GCS) in my Node.js application using Axios client. On develop mode using my PC I pass a Bearer Token and all works properly.

But, I need to use this in production in a cluster hosted on Google Kubernetes Engine (GKE).

I made recommended tuturials to create a service account (GSA), then I vinculed with kubernetes account (KSA), via Workload identity approach, but when I try get files througt one endpoint on my app, I'm receiving:

{"statusCode":401,"message":"Unauthorized"}

What is missing to make?

**Update: What I've done:**

Create Google Service Account

<https://cloud.google.com/iam/docs/creating-managing-service-accounts>

Create Kubernetes Service Account

# gke-access-gcs.ksa.yaml file

apiVersion: v1

kind: ServiceAccount

metadata:

name: gke-access-gcs

kubectl apply -f gke-access-gcs.ksa.yaml

Relate KSAs and GSAs

gcloud iam service-accounts add-iam-policy-binding \

--role roles/iam.workloadIdentityUser \

--member "serviceAccount:cluster\_project.svc.id.goog[k8s\_namespace/ksa\_name]" \

gsa\_name@gsa\_project.iam.gserviceaccount.com

Note the KSA and complete the link between KSA and GSA

kubectl annotate serviceaccount \

--namespace k8s\_namespace \

ksa\_name \

iam.gke.io/gcp-service-account=gsa\_name@gsa\_project.iam.gserviceaccount.com

Set Read and Write role:

gcloud projects add-iam-policy-binding project-id \

--member=serviceAccount:gsa-account@project-id.iam.gserviceaccount.com \

--role=roles/storage.objectAdmin

Test access:

kubectl run -it \

--image google/cloud-sdk:slim \

--serviceaccount ksa-name \

--namespace k8s-namespace \

workload-identity-test

The above command works correctly. Note that was passed --serviceaccount and workload-identity. Is this necessary to GKE?

PS: I don't know if this influences, but I am using SQL Cloud with proxy in the project.

You will need to be authorised to access the files in your Cloud Storage, unless they are publicly available. Previously, you would create a service account and download as json which you can include in a Kubernetes secret ([cloud.google.com/kubernetes-engine/docs/tutorials/…](https://cloud.google.com/kubernetes-engine/docs/tutorials/authenticating-to-cloud-platform#kubectl)) but the recommended approach is currently Workload identity [cloud.google.com/kubernetes-engine/docs/how-to/…](https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity)

–

Issue portrayed in the question is related to the fact that axios client **does not use** the Application Default Credentials (as official Google libraries) mechanism that [Workload Identity](https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity) takes advantage of. The ADC checks:

* If the environment variable GOOGLE\_APPLICATION\_CREDENTIALS is set, ADC uses the service account file that the variable points to.
* If the environment variable GOOGLE\_APPLICATION\_CREDENTIALS isn't set, ADC uses the default service account that Compute Engine, Google Kubernetes Engine, App Engine, Cloud Run, and Cloud Functions provide.

-- [*Cloud.google.com: Authentication: Production*](https://cloud.google.com/docs/authentication/production#auth-cloud-implicit-nodejs)

**This means that axios client will need to fall back to the Bearer token authentication method to authenticate against Google Cloud Storage.**

The authentication with Bearer token is described in the official documentation as following:

**API authentication**

To make requests using OAuth 2.0 to either the Cloud Storage [XML API](https://cloud.google.com/storage/docs/xml-api/overview) or [JSON API](https://cloud.google.com/storage/docs/json_api/v1), include your application's access token in the Authorization header in every request that requires authentication. You can generate an access token from the [OAuth 2.0 Playground](https://developers.google.com/oauthplayground/).

Authorization: Bearer OAUTH2\_TOKEN

The following is an example of a request that lists objects in a bucket.

[JSON API](https://cloud.google.com/storage/docs/authentication#json-api)

Use the [list](https://cloud.google.com/storage/docs/json_api/v1/objects/list) method of the Objects resource.

GET /storage/v1/b/example-bucket/o HTTP/1.1

Host: www.googleapis.com

Authorization: Bearer ya29.AHES6ZRVmB7fkLtd1XTmq6mo0S1wqZZi3-Lh\_s-6Uw7p8vtgSwg

-- [Cloud.google.com: Storage: Docs: Api authentication](https://cloud.google.com/storage/docs/authentication#apiauth)

I've included **basic example** of a code snippet using Axios to query the Cloud Storage (requires $ npm install axios):

const Axios = require('axios');

const config = {

headers: { Authorization: 'Bearer ${OAUTH2\_TOKEN}' }

};

Axios.get(

'https://storage.googleapis.com/storage/v1/b/BUCKET-NAME/o/',

config

).then(

(response) => {

console.log(response.data.items);

},

(err) => {

console.log('Oh no. Something went wrong :(');

// console.log(err) <-- Get the full output!

}

);

I left below example of Workload Identity setup with a node.js official library code snippet as it could be useful to other community members.

Posting this answer as I've managed to use Workload Identity and a simple nodejs app to send and retrieve data from GCP bucket.

I included some bullet points for troubleshooting potential issues.

**Steps:**

* Check if GKE cluster has Workload Identity enabled.
* Check if your Kubernetes service account is associated with your Google Service account.
* Check if example workload is using correct Google Service account when connecting to the API's.
* Check if your Google Service account is having correct permissions to access your bucket.

You can also follow the official documentation:

[*Cloud.google.com: Kubernetes Engine: Workload Identity*](https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity)

Assuming that:

* Project (ID) named: awesome-project <- **it's only example**
* Kubernetes namespace named: bucket-namespace
* Kubernetes service account named: bucket-service-account
* Google service account named: google-bucket-service-account
* Cloud storage bucket named: workload-bucket-example <- **it's only example**

I've included the commands:

$ kubectl create namespace bucket-namespace

$ kubectl create serviceaccount --namespace bucket-namespace bucket-service-account

$ gcloud iam service-accounts create google-bucket-service-account

$ gcloud iam service-accounts add-iam-policy-binding --role roles/iam.workloadIdentityUser --member "serviceAccount:awesome-project.svc.id.goog[bucket-namespace/bucket-service-account]" google-bucket-service-account@awesome-project.iam.gserviceaccount.com

$ kubectl annotate serviceaccount --namespace bucket-namespace bucket-service-account iam.gke.io/gcp-service-account=google-bucket-service-account@awesome-project-ID.iam.gserviceaccount.com

Using the guide linked above check the service account authenticating to API's:

$ kubectl run -it --image google/cloud-sdk:slim --serviceaccount bucket-service-account --namespace bucket-namespace workload-identity-test

The output of $ gcloud auth list should show:

Credentialed Accounts

ACTIVE ACCOUNT

\* google-bucket-service-account@AWESOME-PROJECT.iam.gserviceaccount.com

To set the active account, run:

$ gcloud config set account `ACCOUNT`

Google service account created earlier should be present in the output!

Also it's required to add the permissions for the service account to the bucket. You can either:

* Use Cloud Console
* Run: $ gsutil iam ch serviceAccount:google-bucket-service-account@awesome-project.iam.gserviceaccount.com:roles/storage.admin gs://workload-bucket-example

To download the file from the workload-bucket-example following code can be used:

// Copyright 2020 Google LLC

/\*\*

\* This application demonstrates how to perform basic operations on files with

\* the Google Cloud Storage API.

\*

\* For more information, see the README.md under /storage and the documentation

\* at https://cloud.google.com/storage/docs.

\*/

const path = require('path');

const cwd = path.join(\_\_dirname, '..');

function main(

bucketName = 'workload-bucket-example',

srcFilename = 'hello.txt',

destFilename = path.join(cwd, 'hello.txt')

) {

const {Storage} = require('@google-cloud/storage');

// Creates a client

const storage = new Storage();

async function downloadFile() {

const options = {

// The path to which the file should be downloaded, e.g. "./file.txt"

destination: destFilename,

};

// Downloads the file

await storage.bucket(bucketName).file(srcFilename).download(options);

console.log(

`gs://${bucketName}/${srcFilename} downloaded to ${destFilename}.`

);

}

downloadFile().catch(console.error);

// [END storage\_download\_file]

}

main(...process.argv.slice(2));

The code is exact copy from:

* [*Googleapis.dev: NodeJS: Storage*](https://googleapis.dev/nodejs/storage/latest/)
* [*Github.com: Googleapis: Nodejs-storage: downloadFile.js*](https://github.com/googleapis/nodejs-storage/blob/master/samples/downloadFile.js)

Running this code should produce an output:

root@ubuntu:/# nodejs app.js

gs://workload-bucket-example/hello.txt downloaded to /hello.txt.

root@ubuntu:/# cat hello.txt

Hello there!