Get unlimited access to the best of Medium for less than $1/week. Become a member

Pet Image Viewer: Building a Secure Serverless App with Web Identity Federation using AWS Cognito, CloudFront, S3 & Google OAuth

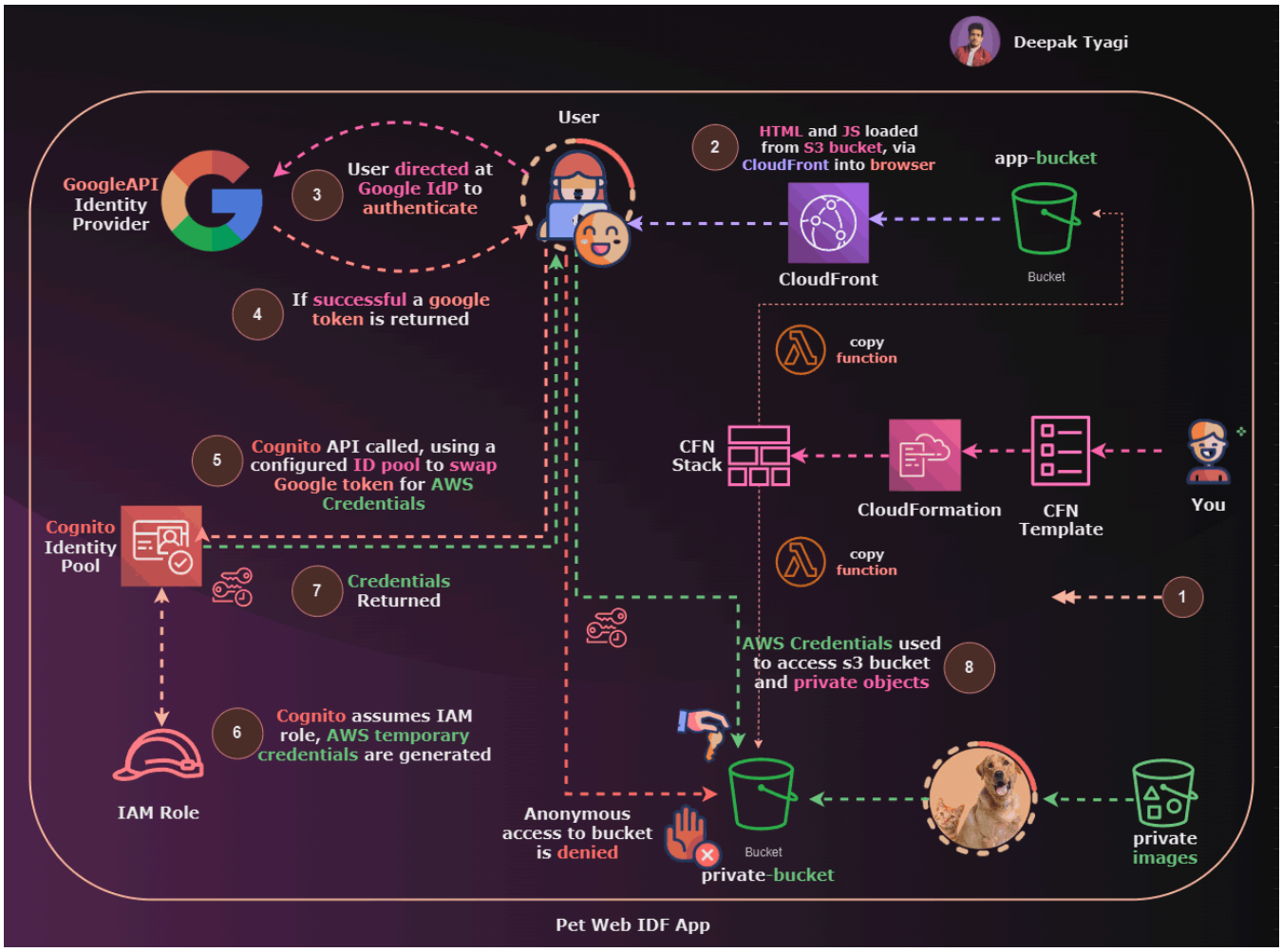
Deepak Tyagi · Follow

16 min read · Nov 10, 2024

Listen Share More

In today's cloud-centric world, managing secure access to resources is a critical skill. This blog post takes you through creating a simple yet secure serverless application that uses Web Identity Federation to authenticate users via Google and grant them controlled access to AWS resources.

From provisioning resources to configuring identity pools and testing the application, this tutorial provides a step-by-step guide. Whether you're experienced developers or just diving into serverless architecture, this project will help you enhance your skills in authentication, identity management, and serverless design.



Architecture Diagram

Table of Contents —

Prerequisites

Before we get started, ensure you have:

- An AWS account

- **Google Account** (Gmail) for setting up Google API credentials.

Project Description:

In this advanced demo, we'll create a serverless web application that uses Google as an identity provider to securely access images in a private S3 bucket.

This application which will be created by the **CloudFormation** template, hosted on **S3 (with CloudFront)**, allows users to log in with their **Google** account, swap their token for AWS credentials using **Cognito** and **IAM**, and view images through pre-signed URLs.

> *Github Repository Link*

Stages:

- **Stage 1:** Set up the environment.
- **Stage 2:** Create a Google API Project.
- **Stage 3:** Configure Cognito Identity Pool.
- **Stage 4:** Update S3 Bucket & Test.
- **Stage 5:** Clean up resources.

Stage 1: Set up the environment.

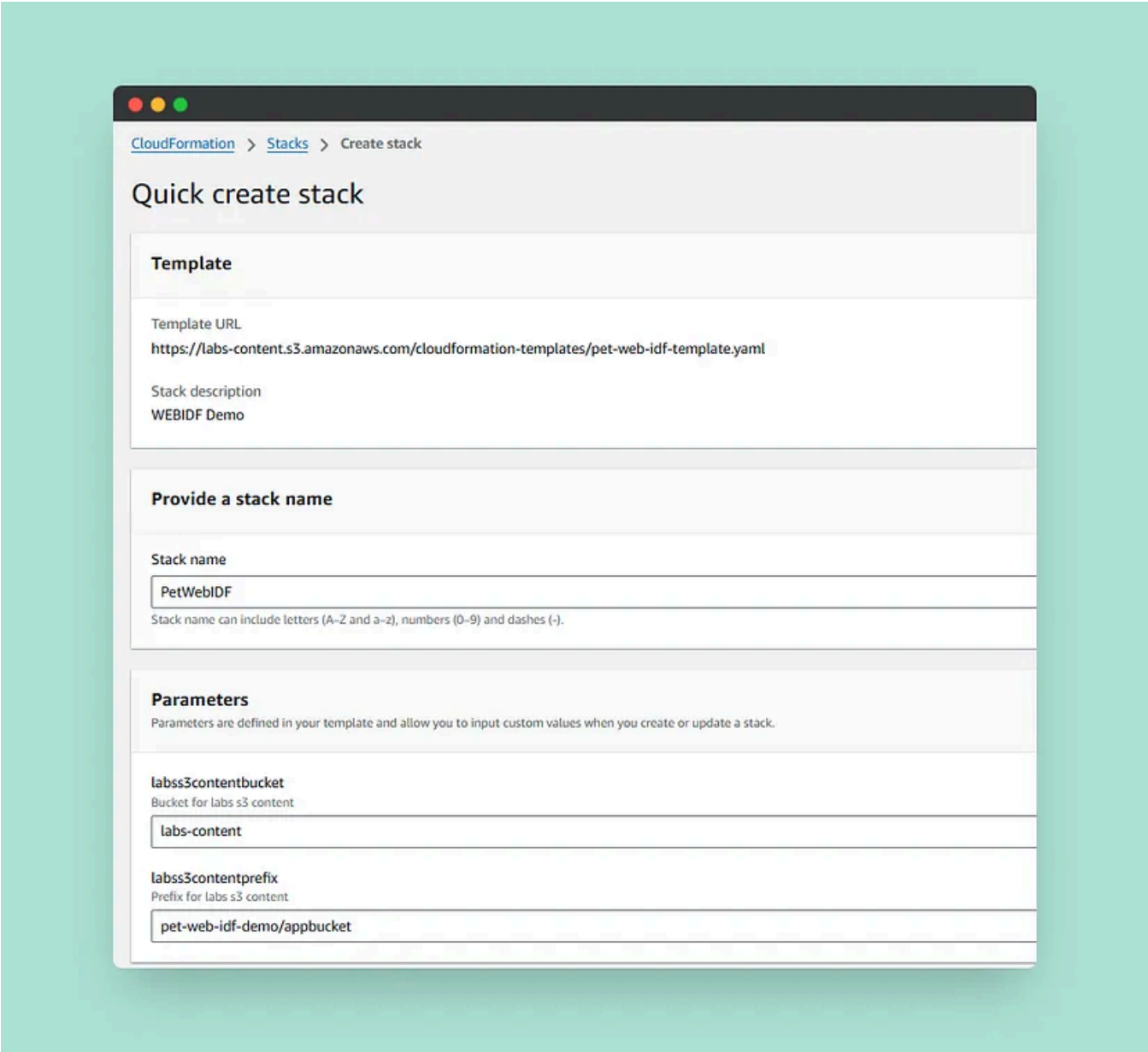
In this stage, we'll walk through the steps to set up the infrastructure for our web application in AWS. This includes logging in to your AWS account, verifying S3 bucket configurations, ensuring your CloudFront distribution is set up correctly, and checking the resources needed for a fully functioning web application environment.

Stage 1a— Login to Your AWS Account

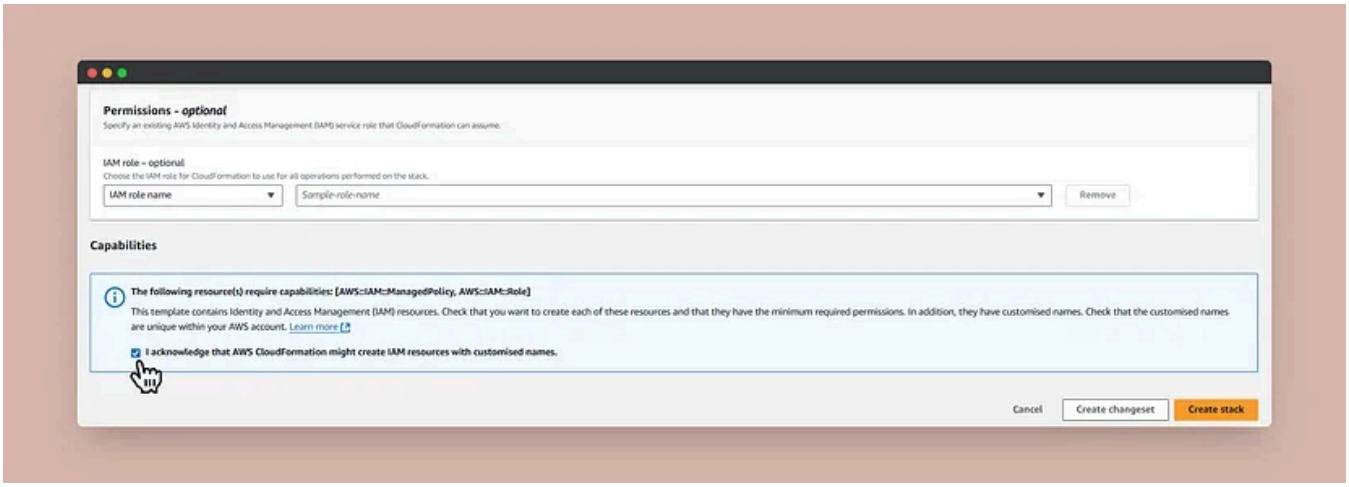
The first step is to log in to your AWS account using a user with admin privileges. It's important to ensure that the region is set to **us-east-1 (N. Virginia)**, as this will be the region where we deploy our resources.

- **Log in to AWS Console:** Open the AWS Management Console, and sign in with your admin account.

- **Set the Region:** Ensure the region is set to **us-east-1 (N. Virginia)**, which is commonly used for deploying web applications.
- **Auto Configure Infrastructure: Quick Setup —** To speed up the setup process, click **HERE** to auto-configure the infrastructure that the application requires.
- You can also check the CloudFormation template **here** in the GitHub Repo.



- **Confirm Resource Capabilities:** When prompted, check the box for the resources that require capabilities. Specifically, look for `[AWS::IAM::ManagedPolicy, AWS::IAM::Role]`.
- **Create the Stack:** After confirming the settings, click **Create Stack** to initiate the creation of the infrastructure resources.

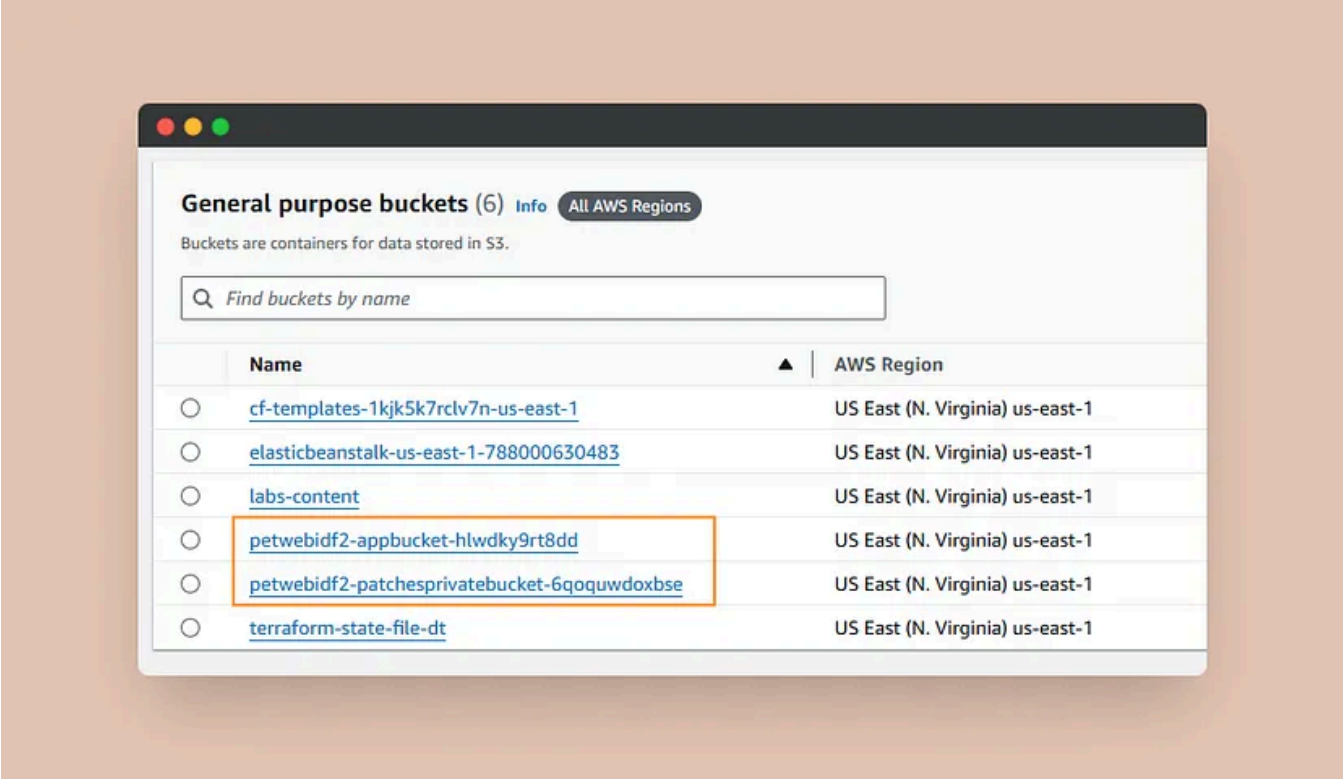


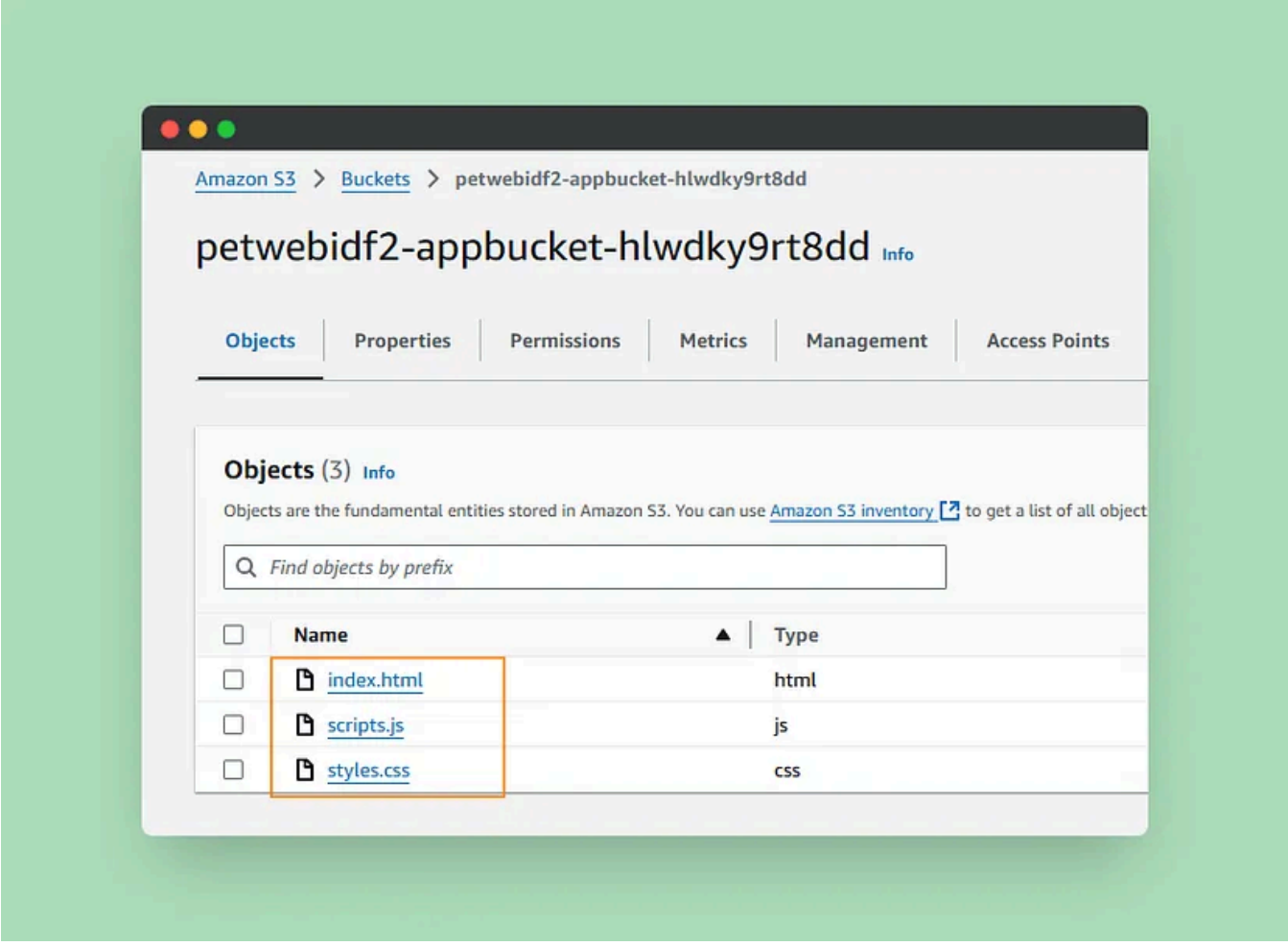
Wait for the stack creation process to complete and ensure it transitions to the **CREATE_COMPLETE** state before proceeding to the next step.

Stage 1b— Verify S3 Bucket

The next step is to verify the **S3 bucket** associated with your application.

1. **Open the S3 Console:** Navigate to the S3 Console.
2. **Locate the Bucket:** Find the bucket starting with `webidf-appbucket`. This is the front-end application bucket.
3. **Verify Bucket Contents:** Open the bucket and ensure it contains objects like `index.html`, `styles.css` and `scripts.js`, which are necessary for your web application to function.
4. **Check Permissions:** Click on the **Permissions** tab in the bucket settings.
5. Ensure **Block all public access** is set to **Off** to allow public access to your web application's assets.
6. **Bucket Policy:** Click on the **Bucket Policy** section to verify that there is an existing bucket policy. This is essential for controlling access to the bucket.

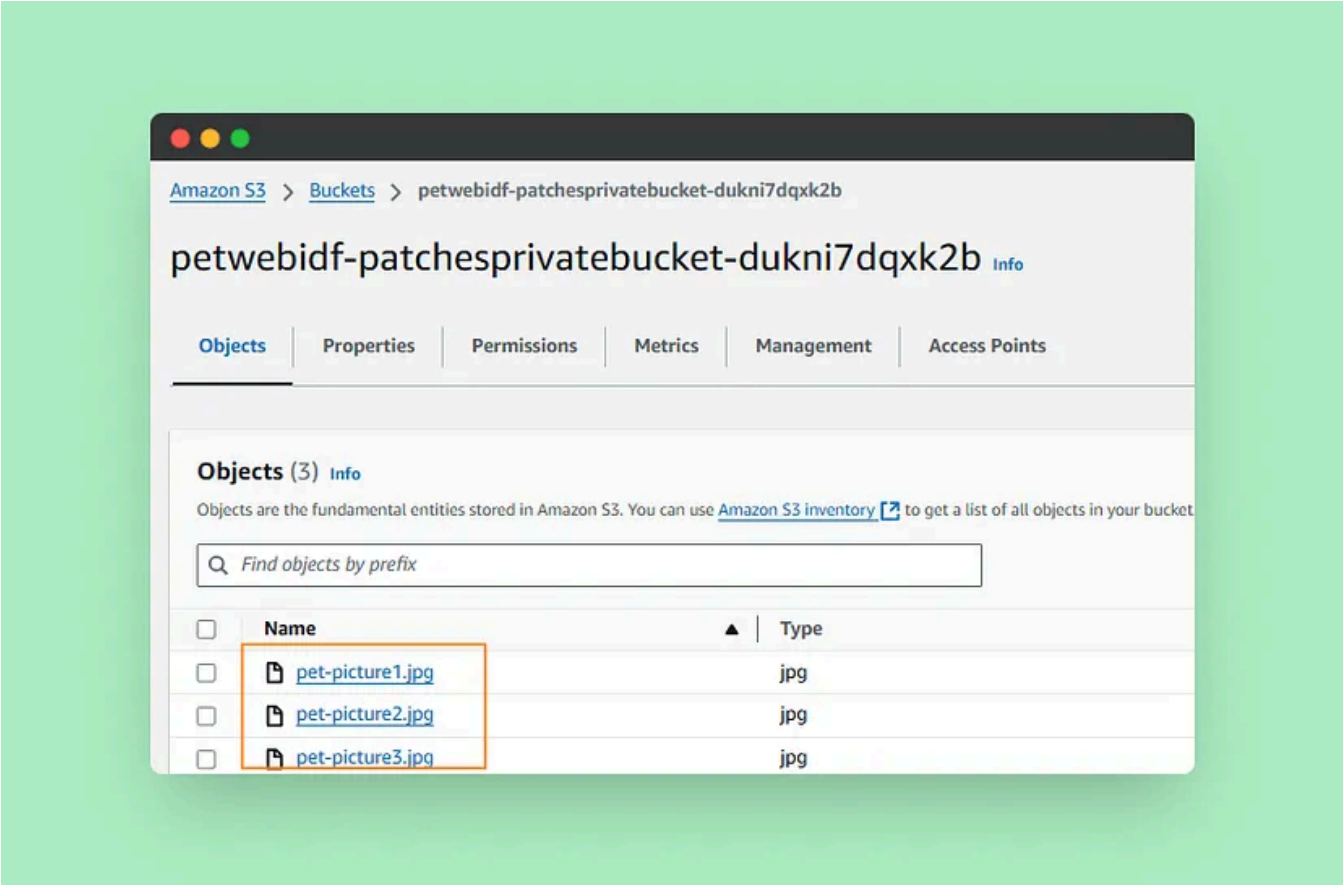




Stage 1c — Verify Private S3 Bucket

The next bucket we need to verify is the **privatebucket**, which holds patches and other private resources.

1. **Locate the Bucket:** Open the bucket starting with `webidf-patchesprivatebucket-`.
2. **Load Objects:** Load the objects in this bucket to familiarize yourself with its contents.
3. **Verify Privacy:** Ensure that this bucket does not have any bucket policy and is entirely private, meaning that only authorized users or applications can access its contents.

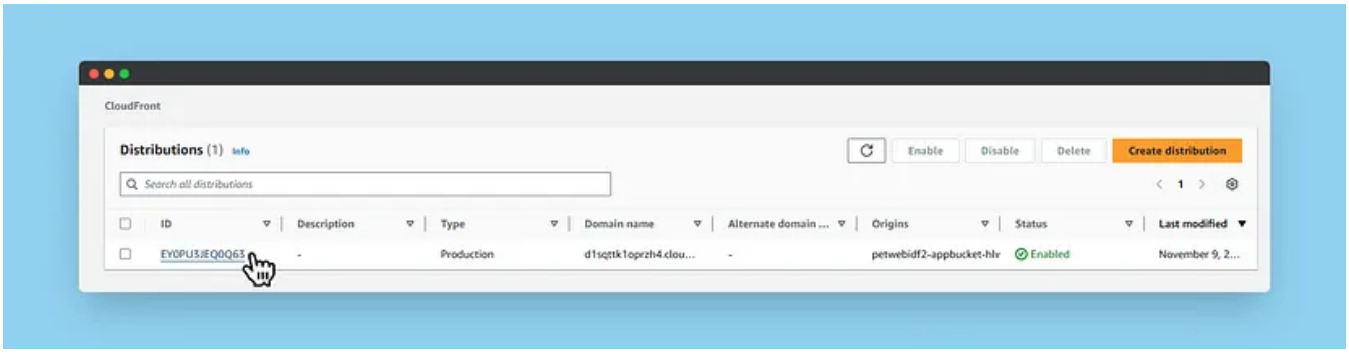


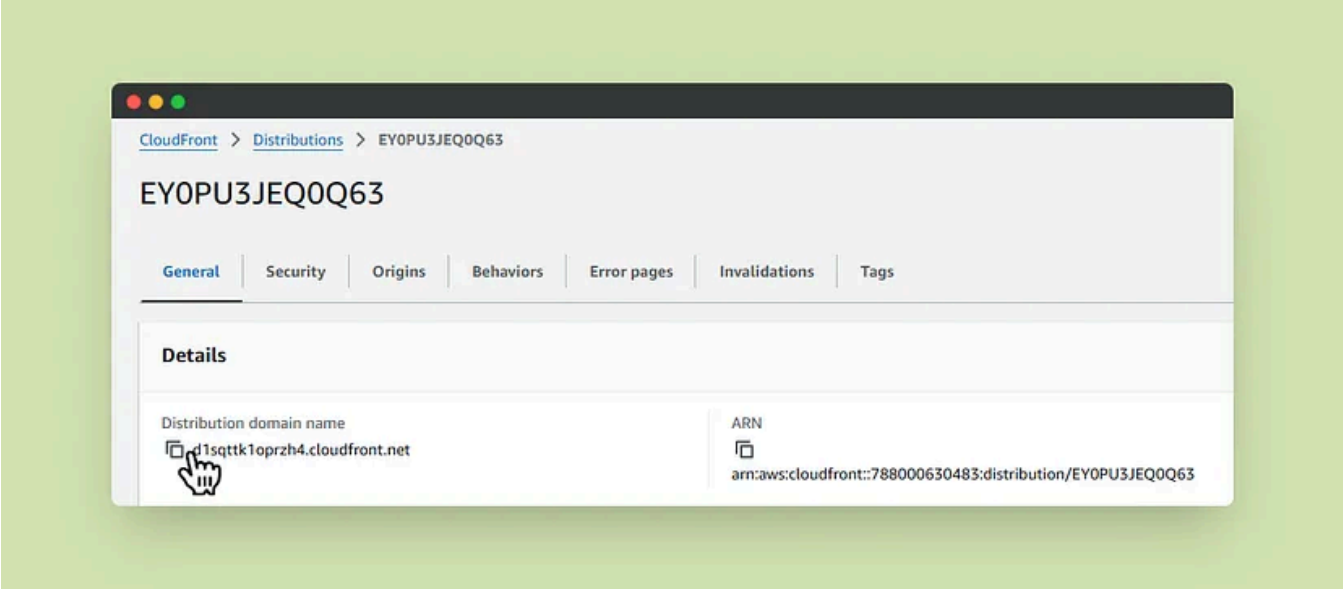
Stage1d — Verify CloudFront Distribution

Now that the S3 buckets are set up, we'll verify the **CloudFront distribution.** CloudFront ensures that your web application is delivered with caching and HTTPS capabilities.

1. **Open CloudFront Console:** Navigate to the CloudFront Console.
2. **Locate the Distribution:** Find the distribution that points to the origin `webidf-appbucket-...` and click on it.
3. **Distribution Domain Name:** Locate the distribution's domain name. This is the URL through which your web application will be accessible.

4. **WebApp URL:** Note down the domain name, and prefix it with **https://** to form the full WebApp URL. For example, if the domain name is `d1o4f0w1ew0exo.cloudfront.net`, your WebApp URL will be: `https://d1o4f0w1ew0exo.cloudfront.net`





Stage 1 — Finish

At this stage, you have successfully set up the following components of your web application infrastructure:

- **Frontend App Bucket:** Stores the web application assets like HTML, CSS and JavaScript files.
- **Private Patches Bucket:** Holds private resources, ensuring they remain secure.
- **CloudFront Distribution:** Provides caching and HTTPS access to ensure fast and secure delivery of your app.

With these resources in place, your web application is ready for further configuration and deployment.

Stage 2 — Setting Up Google API Project for OAuth 2.0 Integration

In this stage, we'll walk through the process of creating a **Google API project**, configuring the **OAuth consent screen**, and generating **OAuth credentials** to allow our application to interact securely with Google APIs. This is crucial for any application that needs to access Google services on behalf of users.

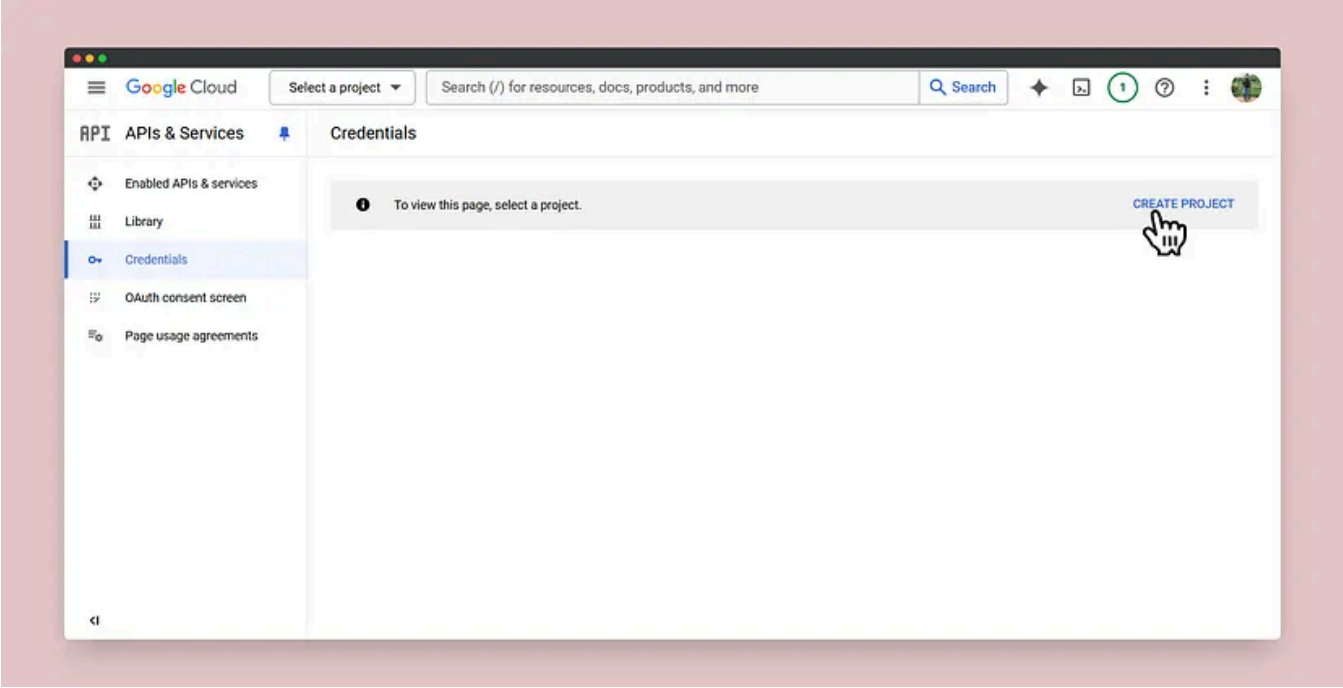
Stage 2a— Create a Google API Project

Before you can use Google APIs with OAuth 2.0 authentication, you need to create an API project in the **Google Cloud Console**. Here's how you can do that:

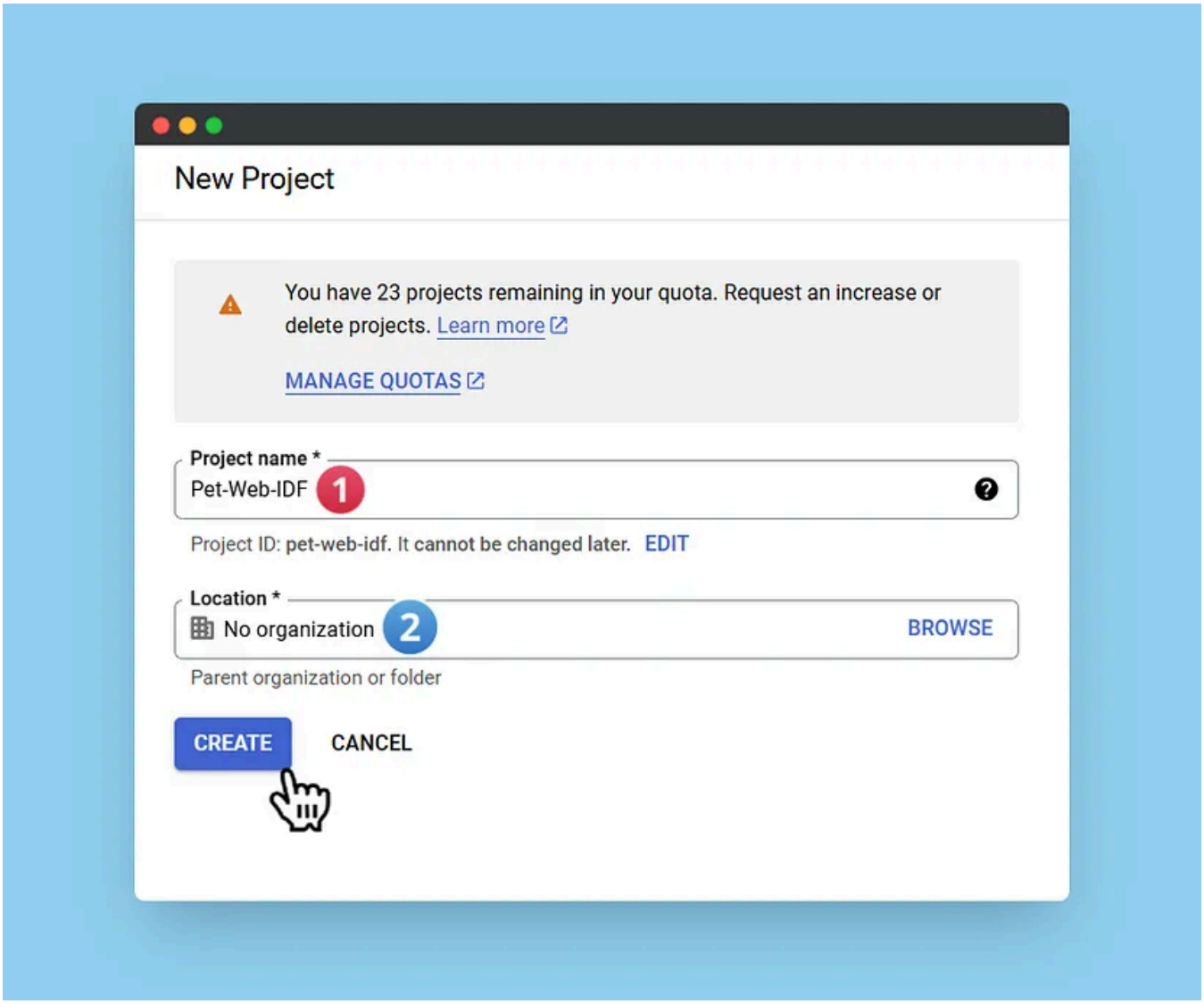
- **Sign in or Create a Google Account:** You'll need a Google account (Gmail will work). If you don't already have one, go ahead and create one.
- **Navigate to Google Credentials Page:** Visit the Google Credentials page in the Google API Console.
- **Sign In or Create Account:** Sign in with your Google account, or if you don't have one, create a new one.
- **Set Up Google API Console:** You may need to set your country and accept terms and conditions. This is a standard process, so feel free to proceed.

1. **Create a New Project:**

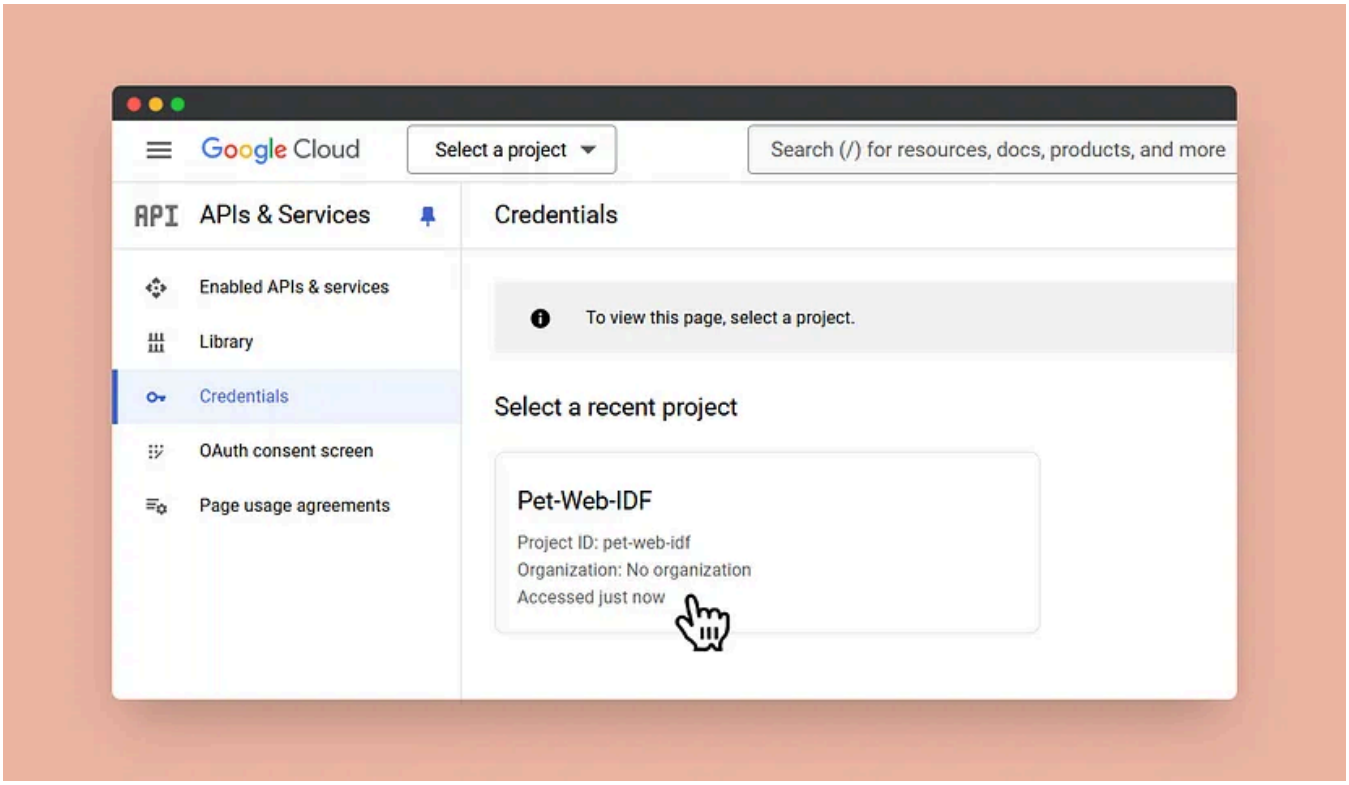
- Click on the **Select a project** dropdown at the top and then click on **NEW PROJECT**.



- Name your project **Pet-Web-IDF.**
- Click **Create** to initialize your project.



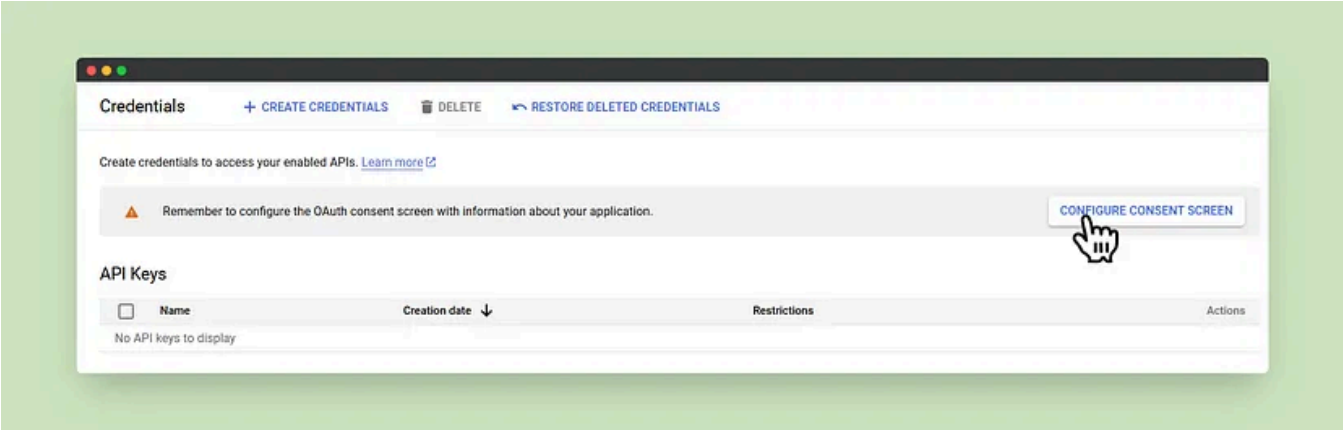
Once the project is created, you'll be able to configure it further to integrate with OAuth 2.0.



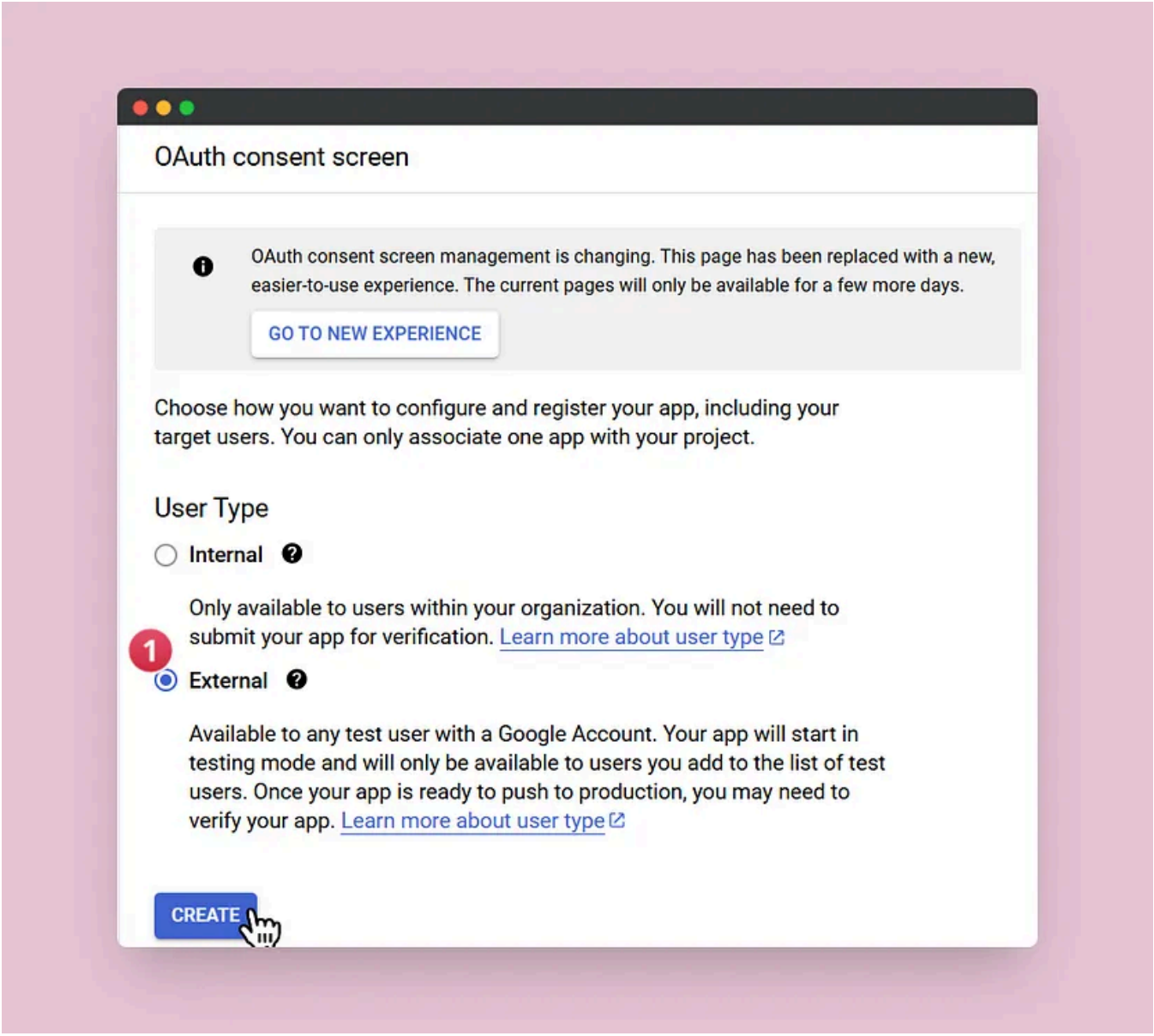
Stage 2b — Configure the OAuth Consent Screen

In this step, you'll configure the **OAuth consent screen.** This is the screen that will be shown to users when your application requests access to their Google account.

- **Go to Credentials:** In the left-hand menu, click on **Credentials**.
- **Configure the Consent Screen:**
- Click **CONFIGURE CONSENT SCREEN.**

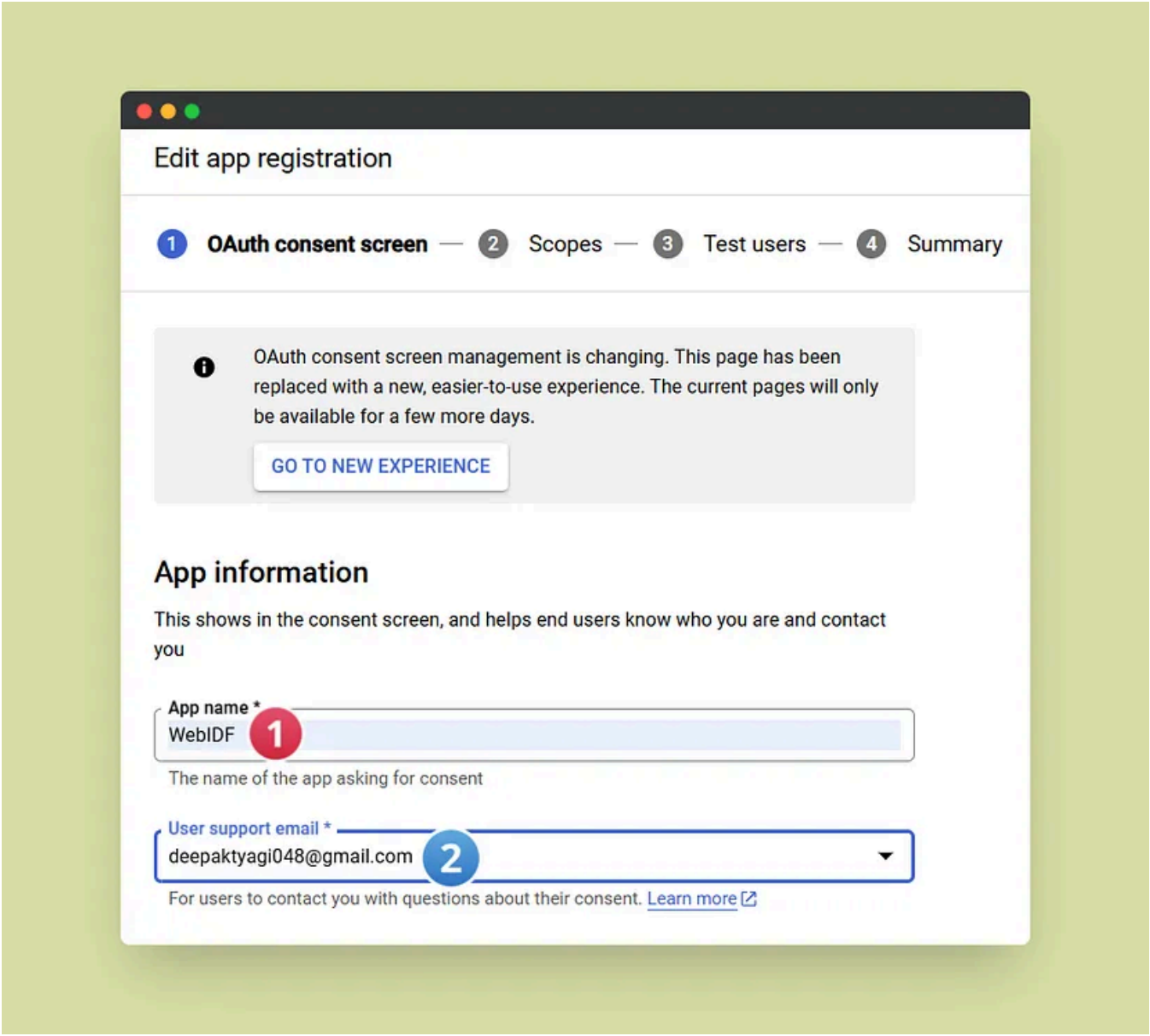


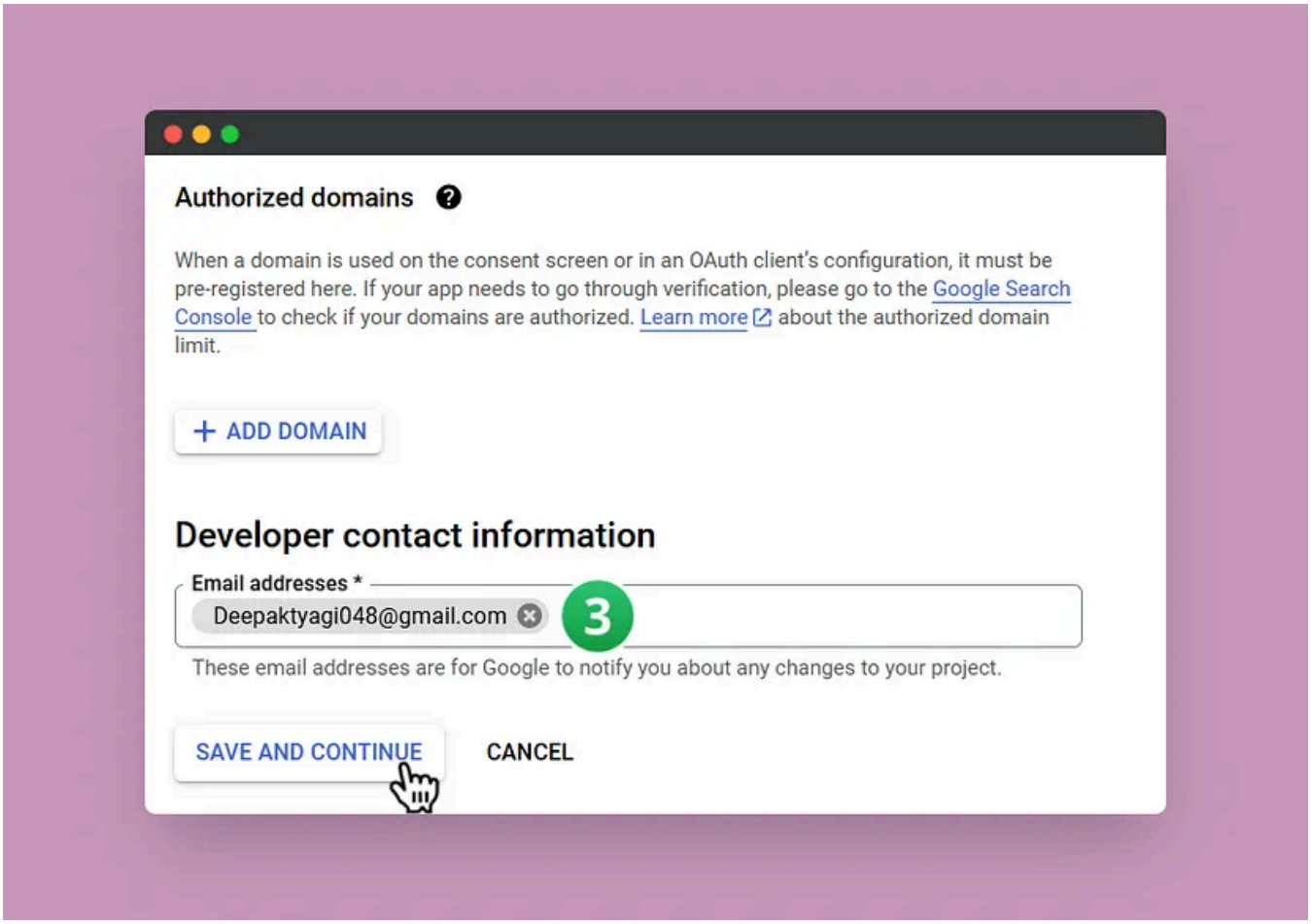
- Since the application will be used by any Google user, select **External** users and click **CREATE.**



1. **Enter Application Details:**

- In the **App Name** field, enter **PetIDF**.
- In the **User Support Email** field, enter your email address.
- In the **Developer Contact Information** field, also enter your email address.


Edit app registration
1 OAuth consent screen — 2 Scopes — 3 Test users — 4 Summary
OAuth consent screen management is changing. This page has been replaced with a new, easier-to-use experience. The current pages will only be available for a few more days.
GO TO NEW EXPERIENCE
App information
This shows in the consent screen, and helps end users know who you are and contact you
App name *
WebIDF
1
The name of the app asking for consent
User support email *
deepaktyagi048@gmail.com
2
For users to contact you with questions about their consent. Learn more



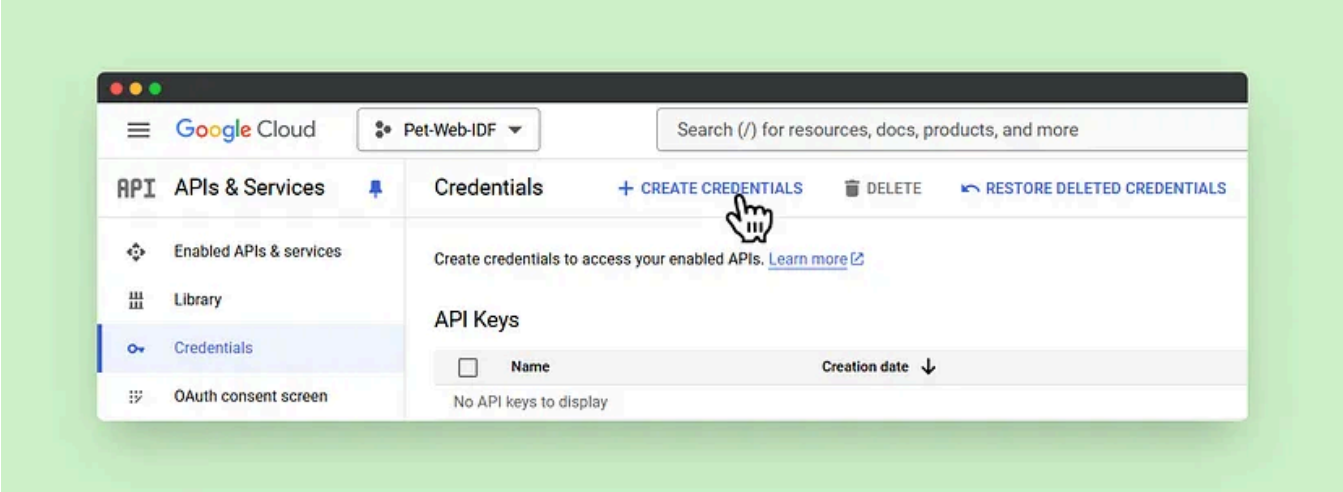
- **Save and Continue:**
- Click **SAVE AND CONTINUE** to move through the next few screens.
- When prompted, click **BACK TO DASHBOARD** to return to the main screen after completing the consent screen setup.

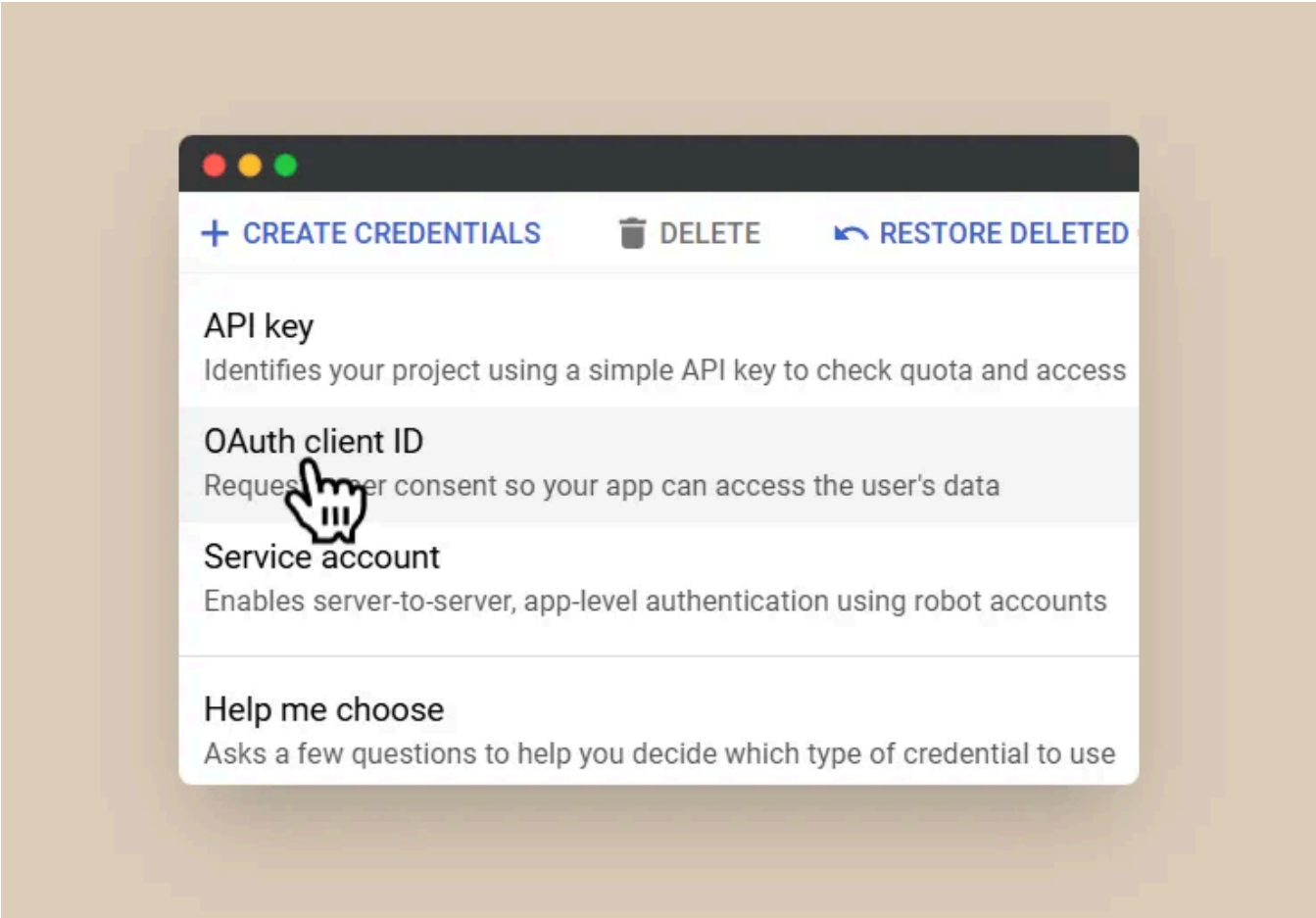
Stage 2c — Create Google API Project Credentials

Now, we'll create the **OAuth credentials** necessary to allow our application to authenticate with Google APIs.

1. **Navigate to Credentials:** In the left menu, click on **Credentials** again.
2. **Create OAuth Client ID:**

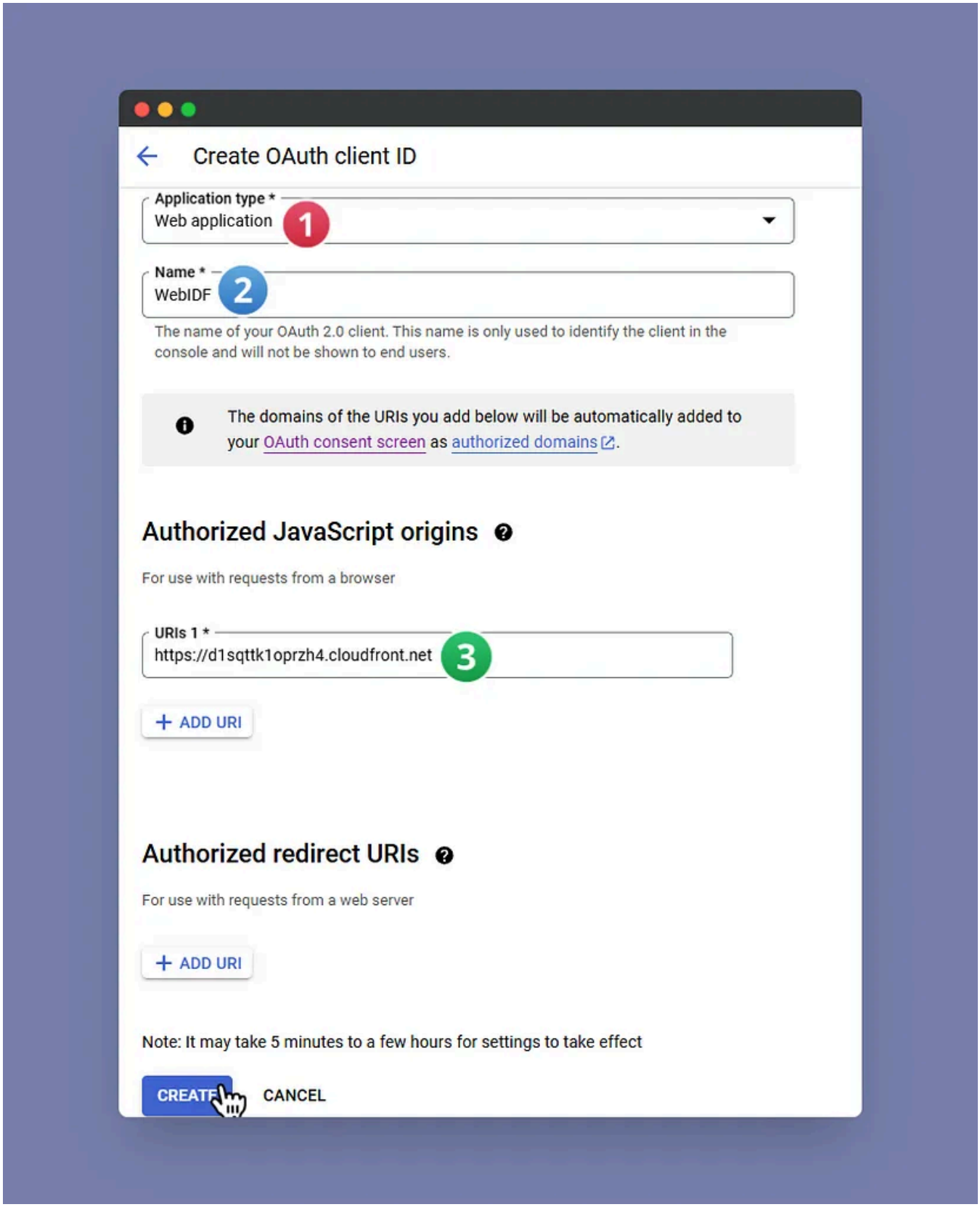
- Click **CREATE CREDENTIALS**, and then select **OAuth client ID**.





- For the **Application type**, select **Web Application**.
- Under **Name**, enter **WebIDF**.
- Add **Authorized JavaScript Origins.**
- You'll need to add the **WebApp URL**, which is the **CloudFront distribution domain** from the deployment process. Make sure it starts with `https://`.
- Click **ADD URI** under **Authorized JavaScript origins**, and enter your distribution's DNS name. It will look something like this:

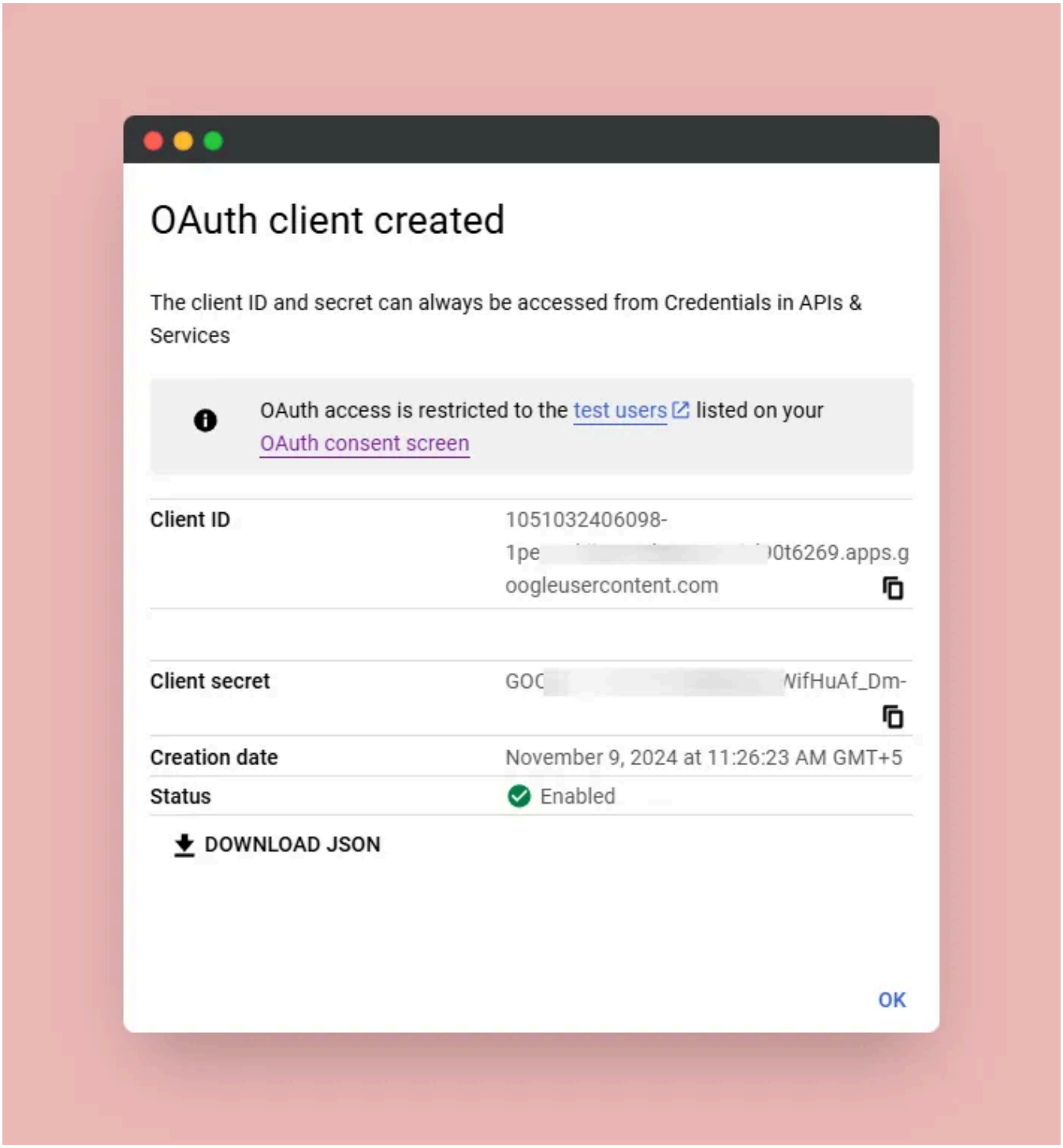
`https://d38sv1tnkmk8i6.cloudfront.net`, but be sure to use your own distribution's DNS name (do not use the example above).



1. **Create OAuth Credentials:** Once you've entered the information, click **CREATE**.

After the credentials are created, you will be presented with two important pieces of information:

- **Client ID:** This is essential for integrating Google APIs with your application.
- **Client Secret:** This will not be needed again, so make sure to keep it secure.



Note down the **Client ID** as you will need it for the next steps. Once you've written it down, click **OK** to complete the process.

Stage 2 — Finish

At this point, we have completed the following steps for integrating Google APIs with our web application:

- **Google API Project Created**: We now have a project set up in the Google Cloud Console.
- **OAuth Consent Screen Configured**: The consent screen is ready for external users to authenticate with our application.
- **OAuth Credentials Generated**: We have the necessary credentials (Client ID) to interact with Google APIs securely.

With these configurations in place, our application is now ready to use Google's OAuth 2.0 authentication for accessing Google APIs.

Stage 3 — Setting Up AWS Cognito Identity Pool and Permissions for Serverless Application Access

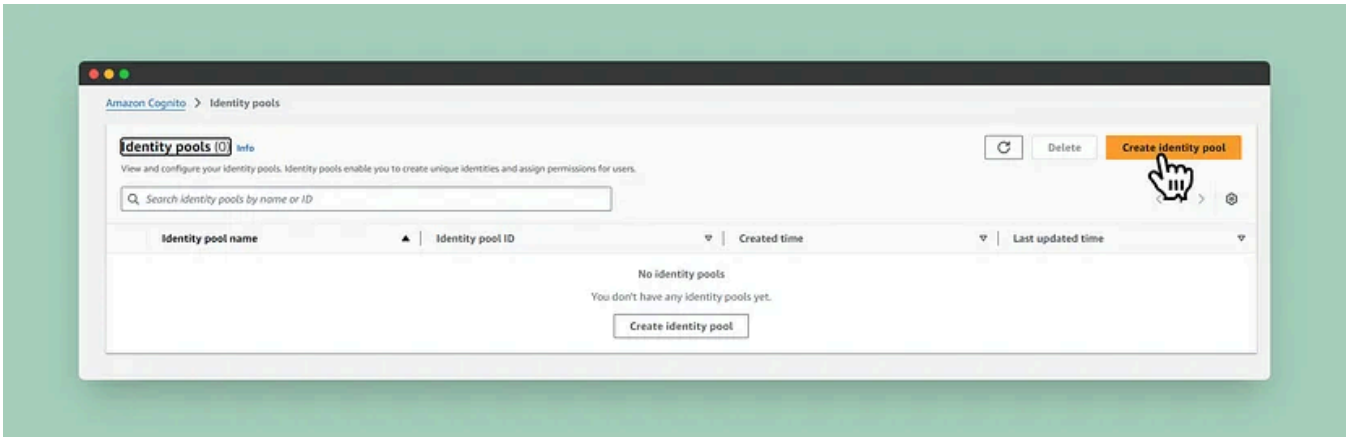
In this section, we'll be setting up an **AWS Cognito Identity Pool**, configuring **IAM roles**, and adjusting permissions to allow secure access to resources. This will enable our serverless application to interact with AWS resources, such as S3 buckets, using temporary credentials.

Stage 3a — Create a Cognito Identity Pool

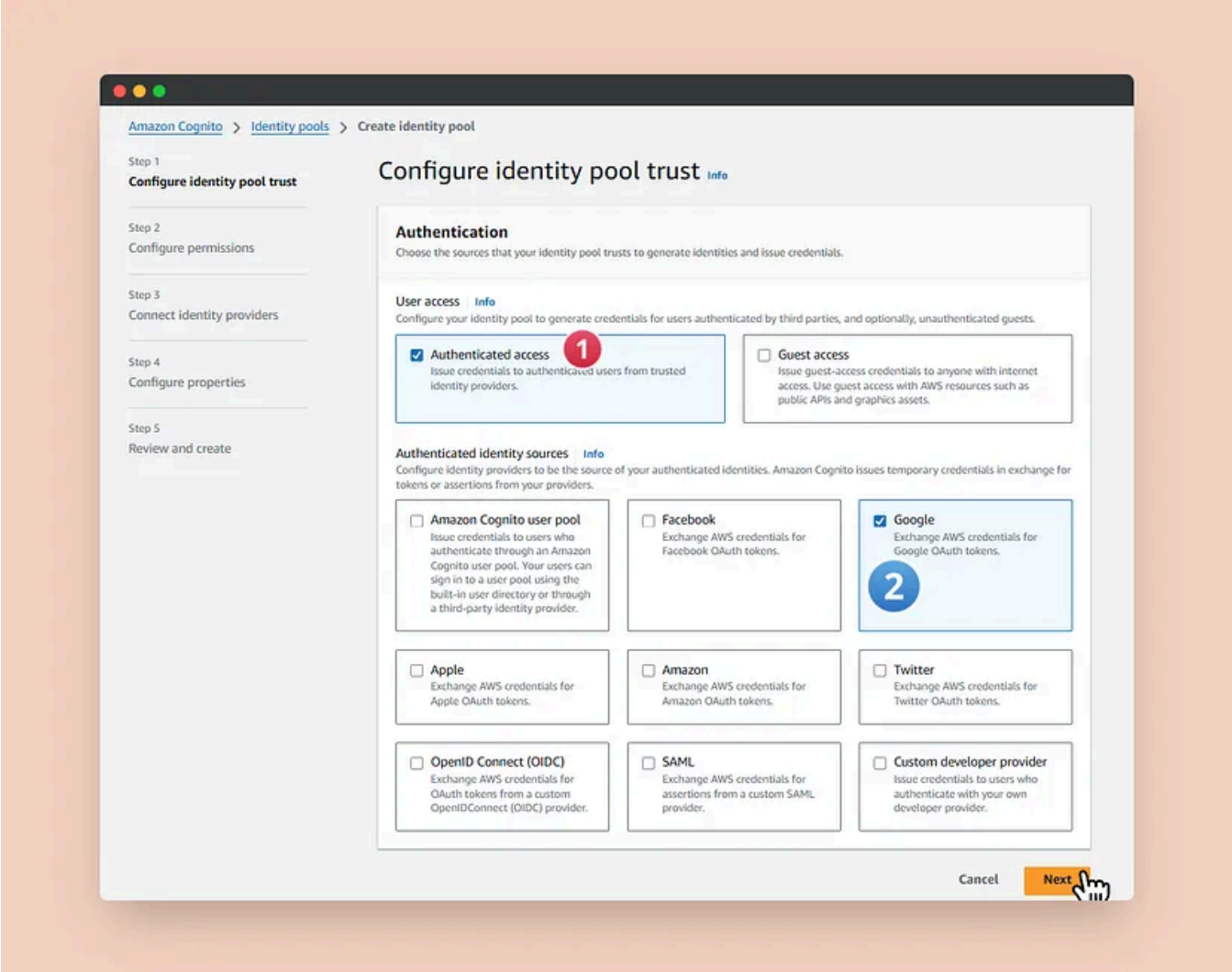
The first step in this stage is to create a Cognito Identity Pool in AWS to manage user identities for accessing your serverless application.

1. **Open the Cognito Console:** Go to the AWS Cognito Console.
2. **Select Federated Identities:** In the menu on the left, choose **Federated Identities**.
3. **Create a New Identity Pool:**

- If this is your first identity pool, the creation process will start automatically.
- If you already have identity pools, click **Federated Identities** and then select **Create new identity pool**.

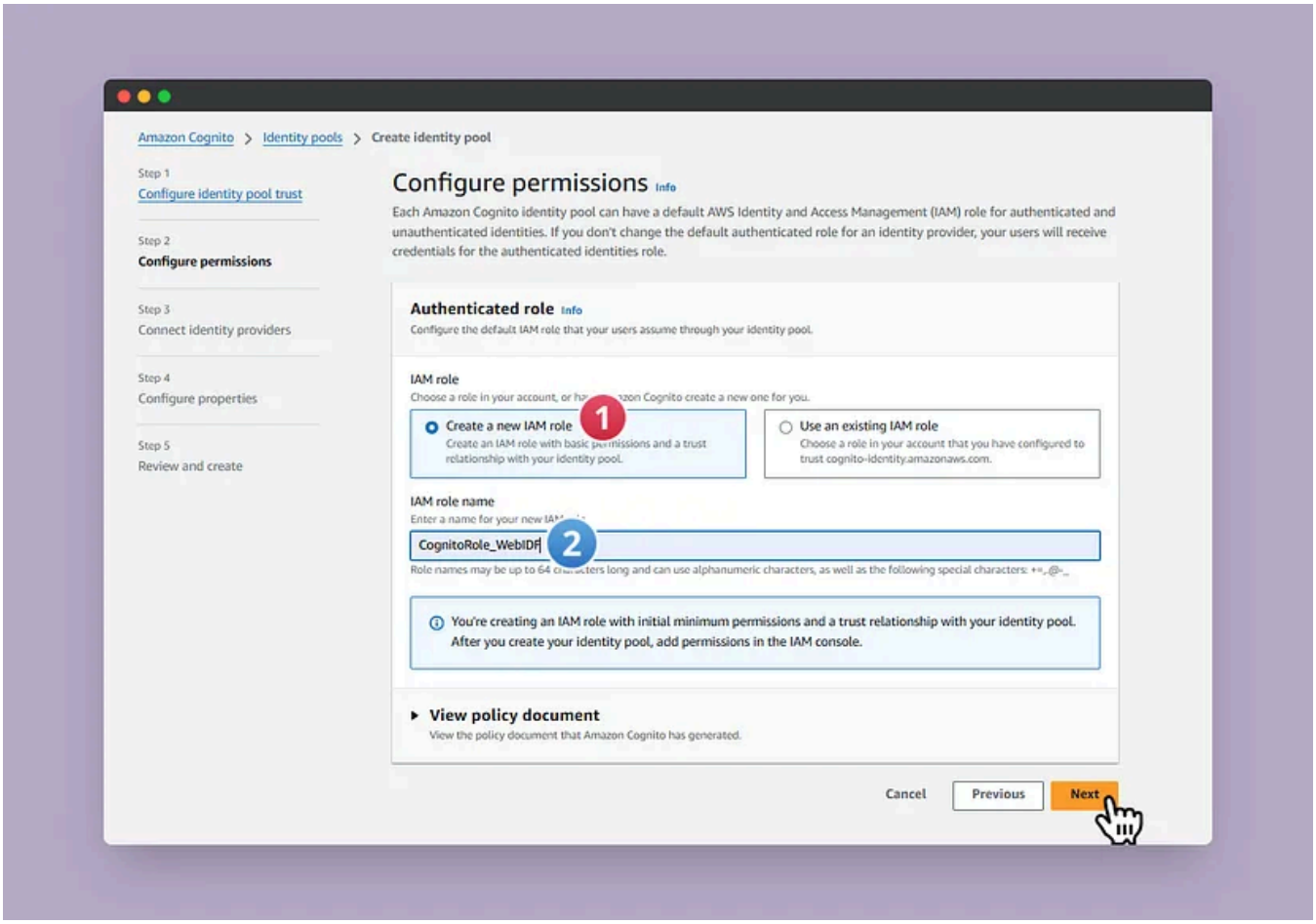


- Select **Authenticated Access** for User Access.
- Add **Google** as an **Authentication Provider:**
- Click on **Next.**

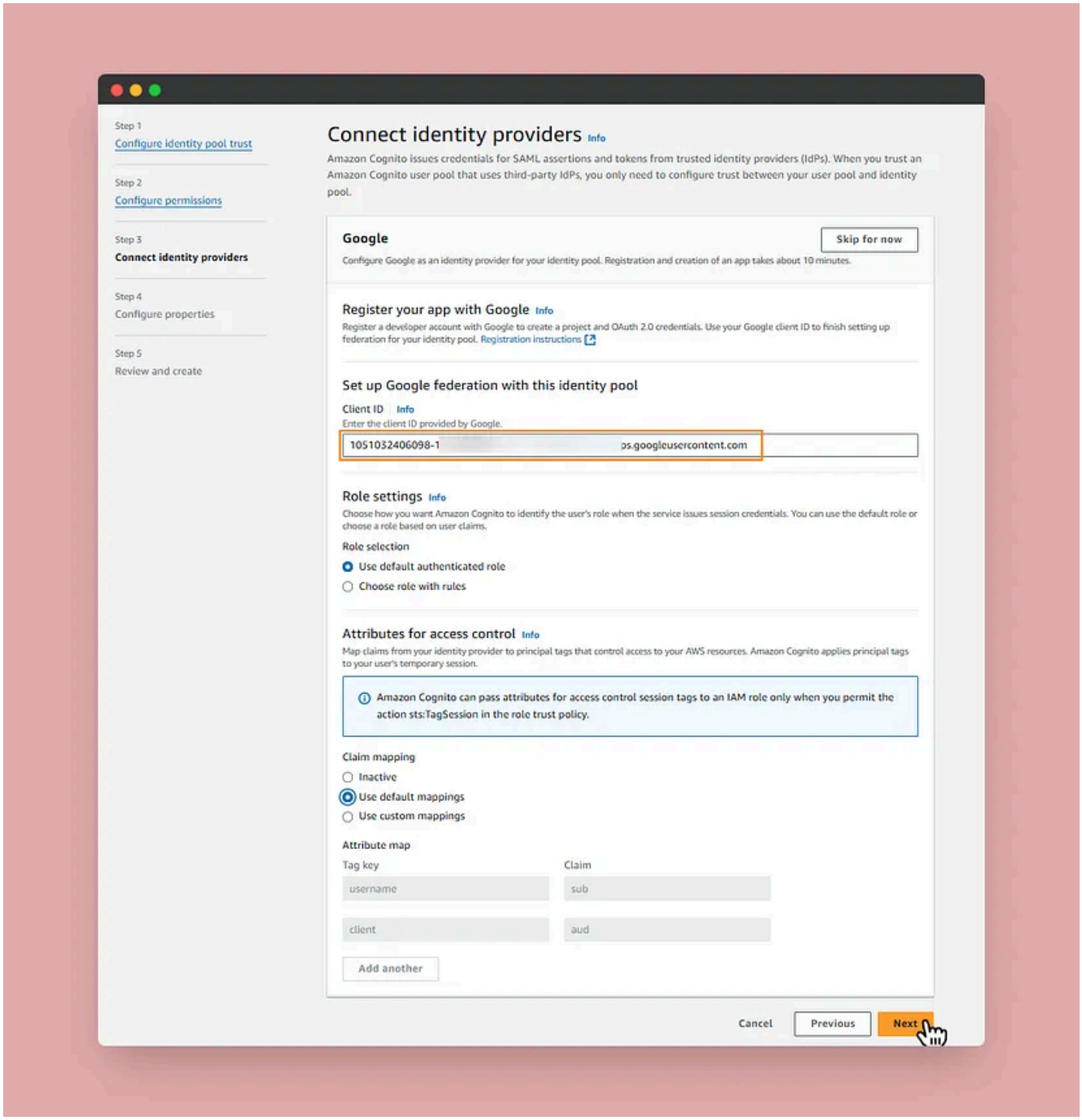


- Now, Select **Create a new Role** and give the role a name `CognitoRole_WebIDF`.

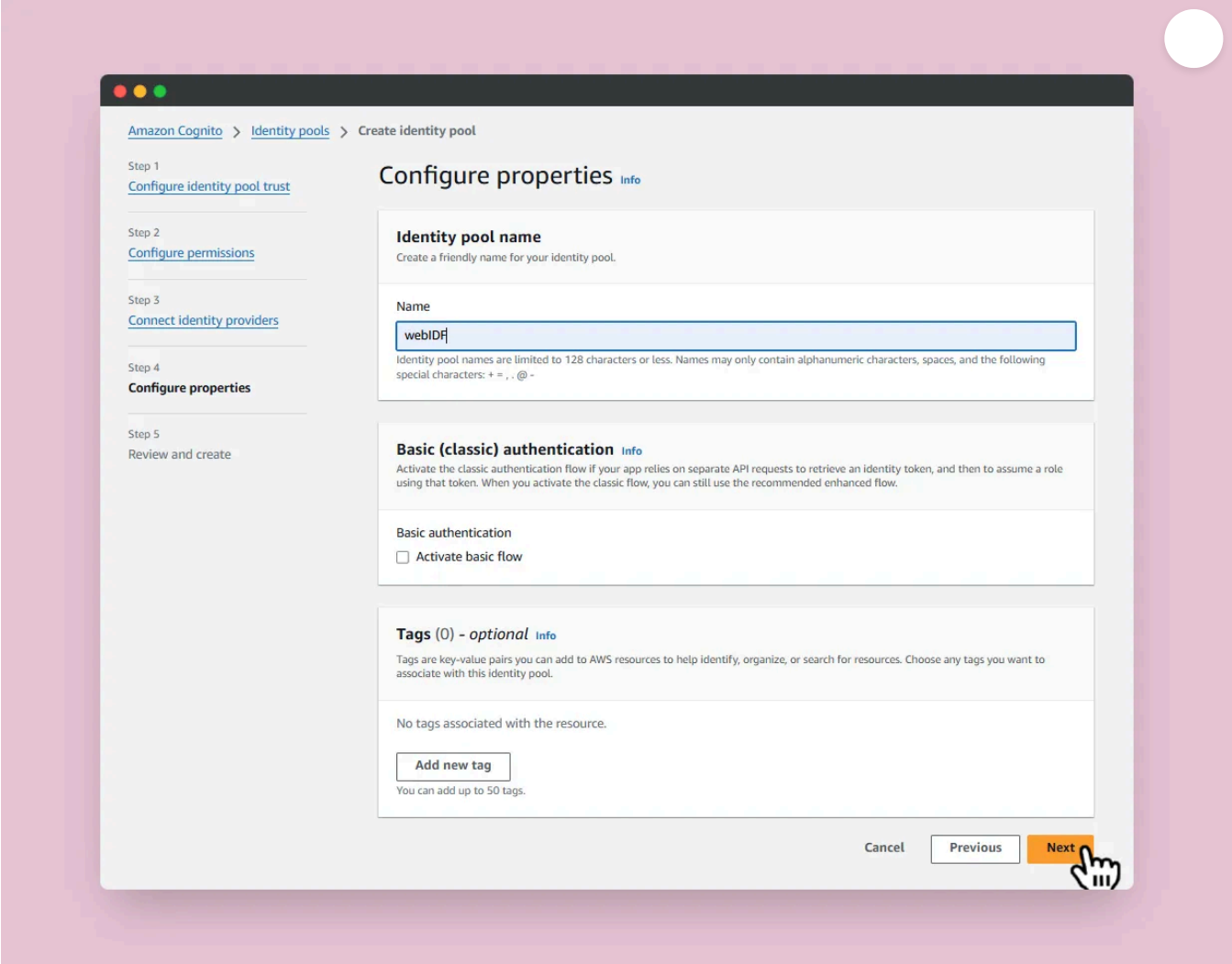
This role will be assumed by the federated identity to grant the user temporary access to AWS resources.



- In the **Google Client ID** box, enter the **Google Client ID** you saved in the previous step.
- Select **Use default mappings** for **Claim mapping.**

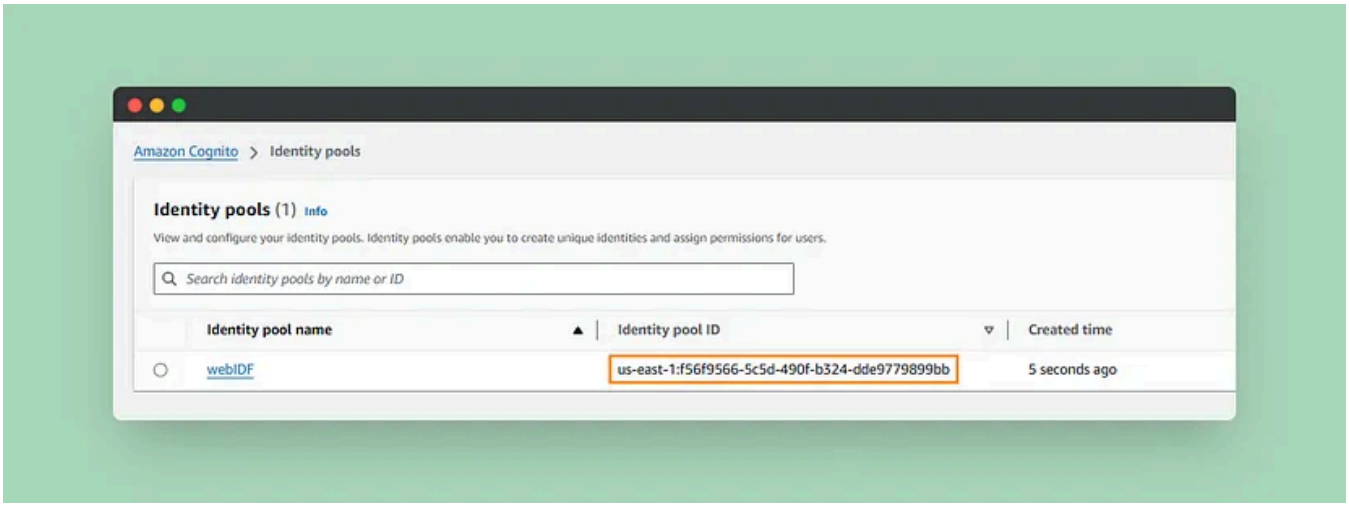


- Give the pool name — `WebIDF`



- **Create the Pool:** Click **Create Pool** to complete the identity pool creation.

Your identity pool is now set up to manage user authentication with Google as a provider.

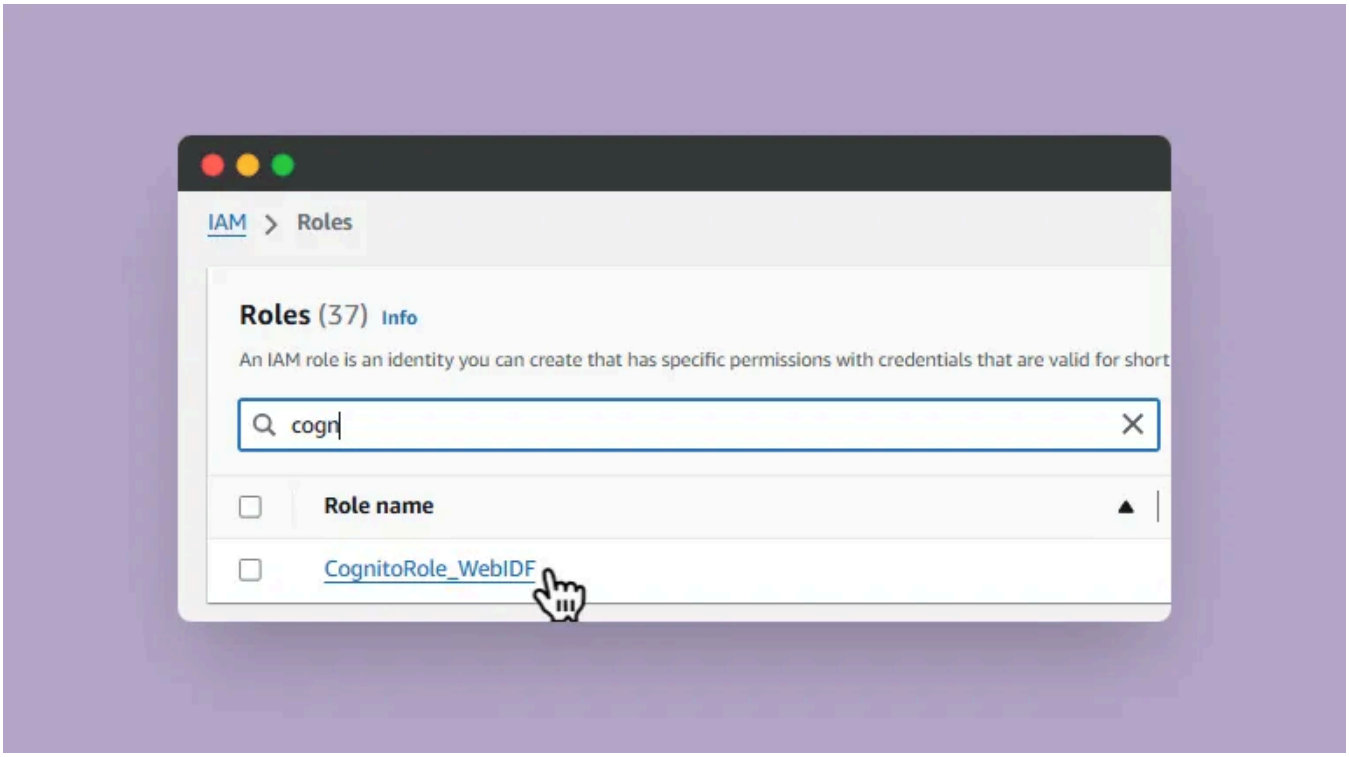


You'll receive an **Identity Pool ID.** Be sure to note down the Identity Pool ID as you'll need it for further configurations.

Stage 3b— Adjust Permissions for Accessing the Private Bucket

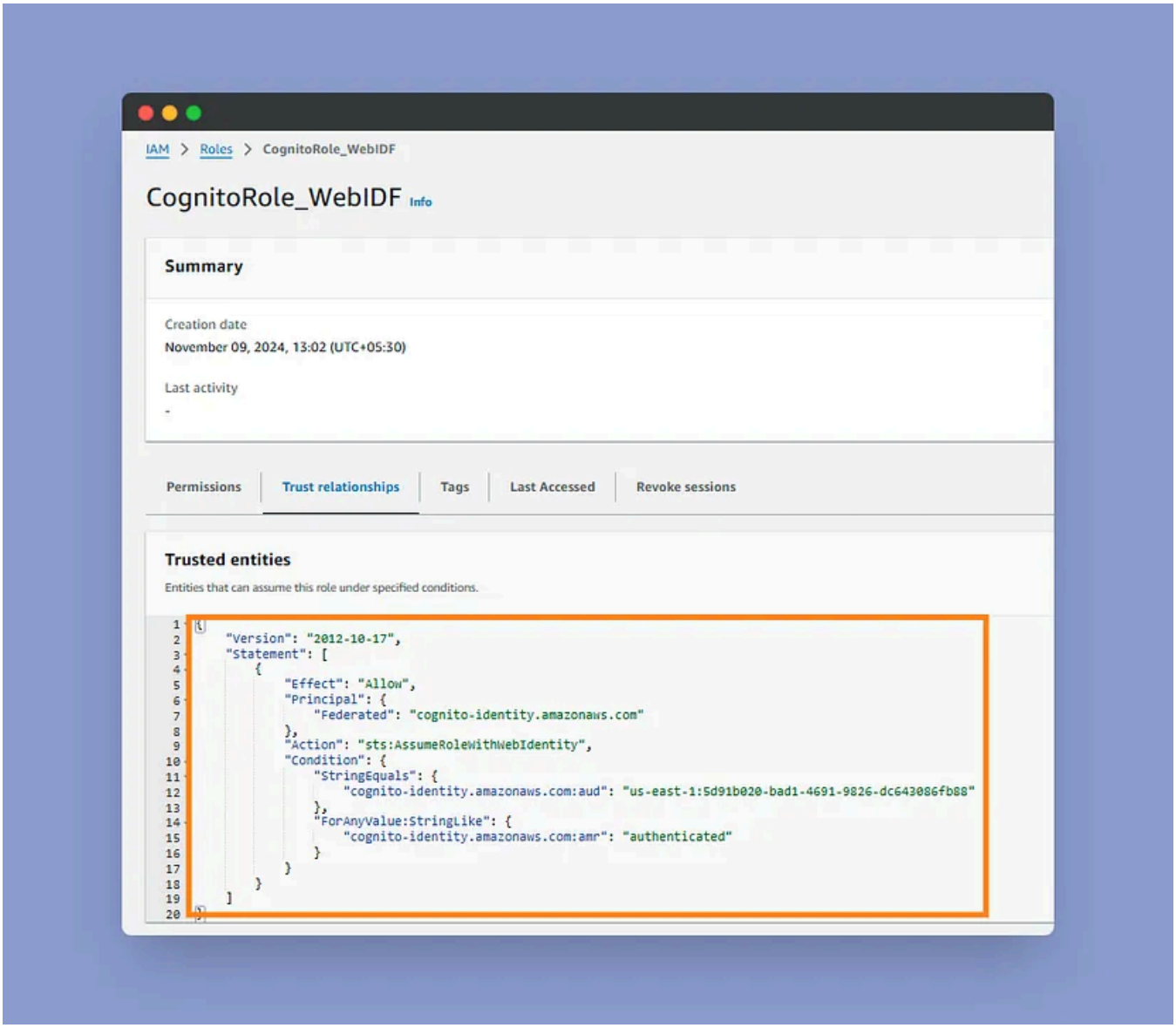
Our serverless application will need permission to read images from a private S3 bucket created by the CloudFormation template. Here's how to configure the necessary permissions.

1. **Open the IAM Console:** Go to the IAM Console.
2. **Locate the Authenticated Role:** Click **Roles** in the left menu and find the role named **CognitoRole_WebIDF**.



3. **Check Trust Relationships:**

- Click **Trust Relationships** in the role's details.
- Verify that the role is assumable by `cognito-identity.amazonaws.com`, with the following conditions:
- **StringEquals** for `cognito-identity.amazonaws.com:aud` should match your Cognito Identity Pool ID.
- **ForAnyValue**for `cognito-identity.amazonaws.com:amr` should be **authenticated.**



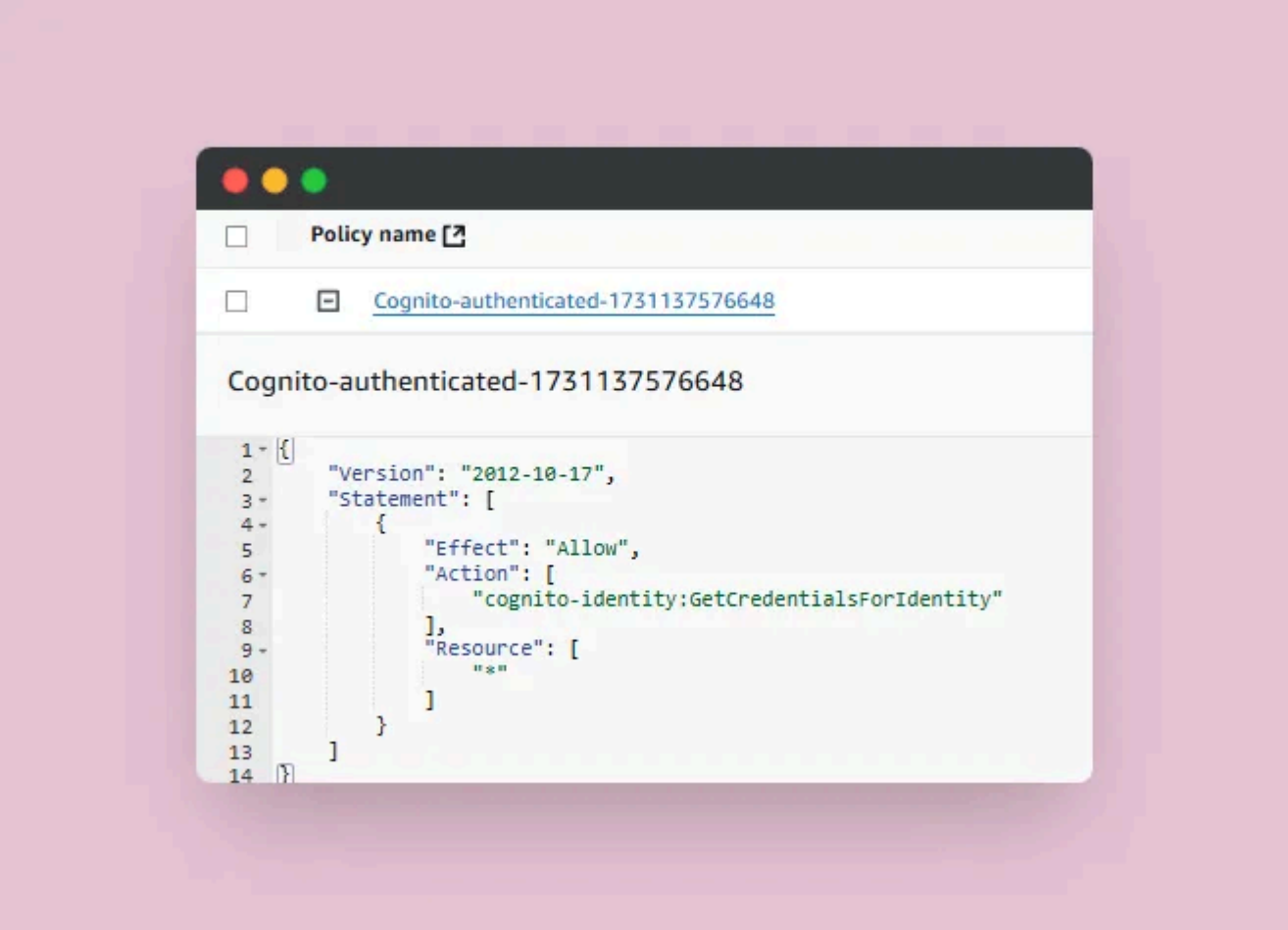
- This configuration ensures that only authenticated users from your identity pool can assume this role. AWS Cognito will manage the temporary credentials for your application, allowing secure access to AWS resources.
- *Optional* — If you want only particular accounts to access your AWS resources, then you can use below policy. Make sure to update the Cognito Identity Pool ID and Gmail account. This is not in the scope of this project as we're allowing access to all external users with Google account.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "cognito-identity.amazonaws.com"
            },
```

```
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "cognito-identity.amazonaws.com:aud": "YOUR_COGNITO_IDENTITY_P
                },
                "StringLike": {
                    "cognito-identity.amazonaws.com:amr": "google",
                    "cognito-identity.amazonaws.com:sub": "accounts.google.com:YOU
                }
            }
        }
    ]
}
```

Define Permissions:

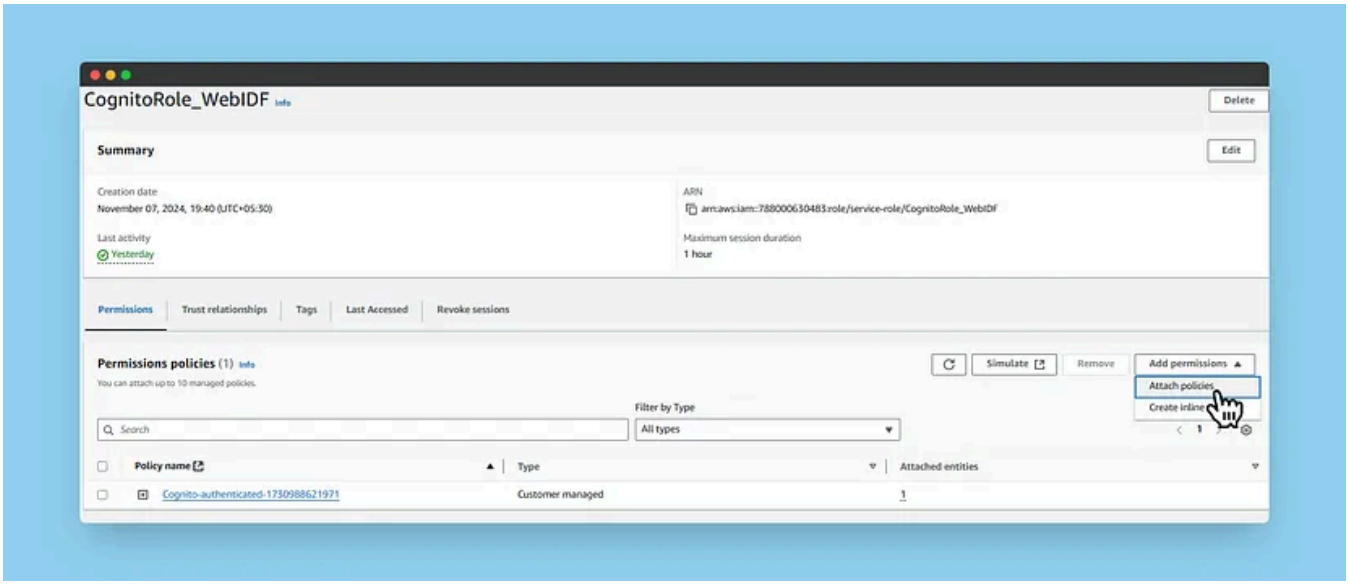
- Click on **Permissions** to see what actions this role is authorized to perform.



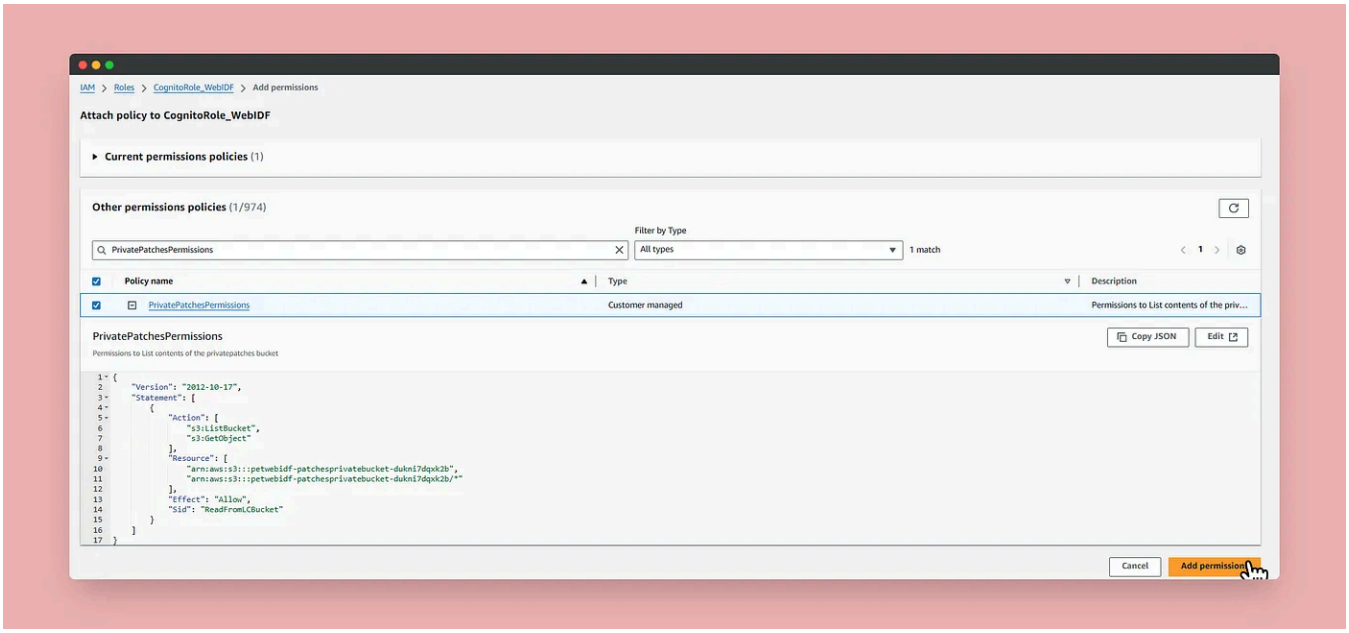
Attach the Policy for Private Bucket Access:

- The CloudFormation template has created a managed policy allowing access to the **privatepatches bucket.**

- Click **Add permissions** and then select **Attach policies.**



- In the search box, type **PrivatePatches** and press **Enter.**
- Check the box next to **PrivatePatchesPermissions** and click **Attach Policies.**



Stage 3 — Finish

With these configurations in place, your infrastructure now includes the following components:

- **Frontend App Bucket:** The S3 bucket holding the frontend application.
- **Configured Google API Project:** Allows OAuth 2.0 authentication with Google.
- **Cognito Identity Pool:** Manages user identities and authentication.

- **IAM Roles for Identity Pool:** Controls access for authenticated and unauthenticated identities.

This completes the setup of the identity pool and the necessary permissions to access AWS resources securely. Your serverless application is now ready to authenticate users, manage identities, and interact with AWS services like S3 through the permissions granted by AWS Cognito.

Stage 4 — Setting Up and Testing our Serverless Application on AWS

In this stage, we'll download and update the HTML and JavaScript files, configure them with our specific credentials, and upload them back to Amazon S3. Then, we'll test the setup to verify the application's connection to Google and AWS resources.

Stage 4a — Download HTML and JavaScript Files from S3

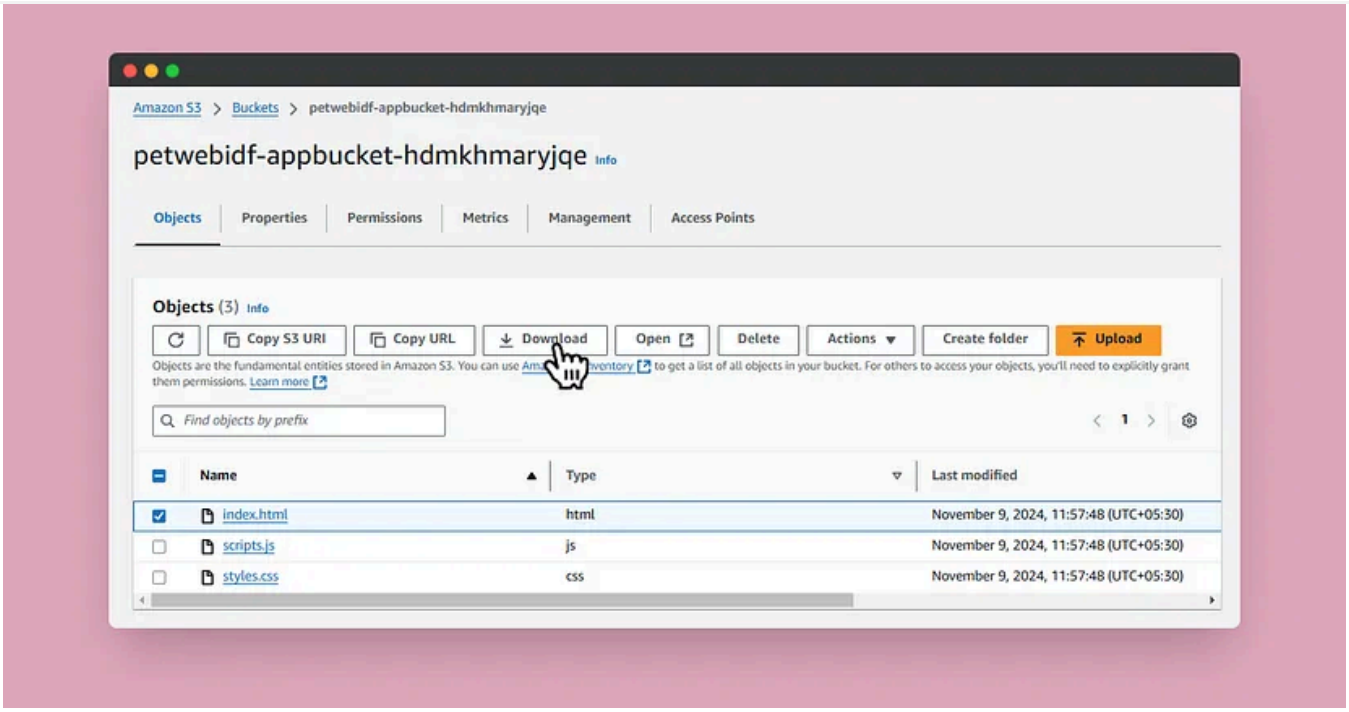
The first step is to download the necessary files from the S3 bucket.

1. **Open the S3 Console:** Go to the AWS S3 Console.
2. **Locate Your Bucket:** Open the bucket named **webidf-appbucket-**.
3. **Download the Files:**

- Select **index.html** and click **Download** to save it locally.
- Select **scripts.js** and click **Download** to save it locally as well.

Open in app ↗

Medium Search



These files will be edited to include your specific connection information.

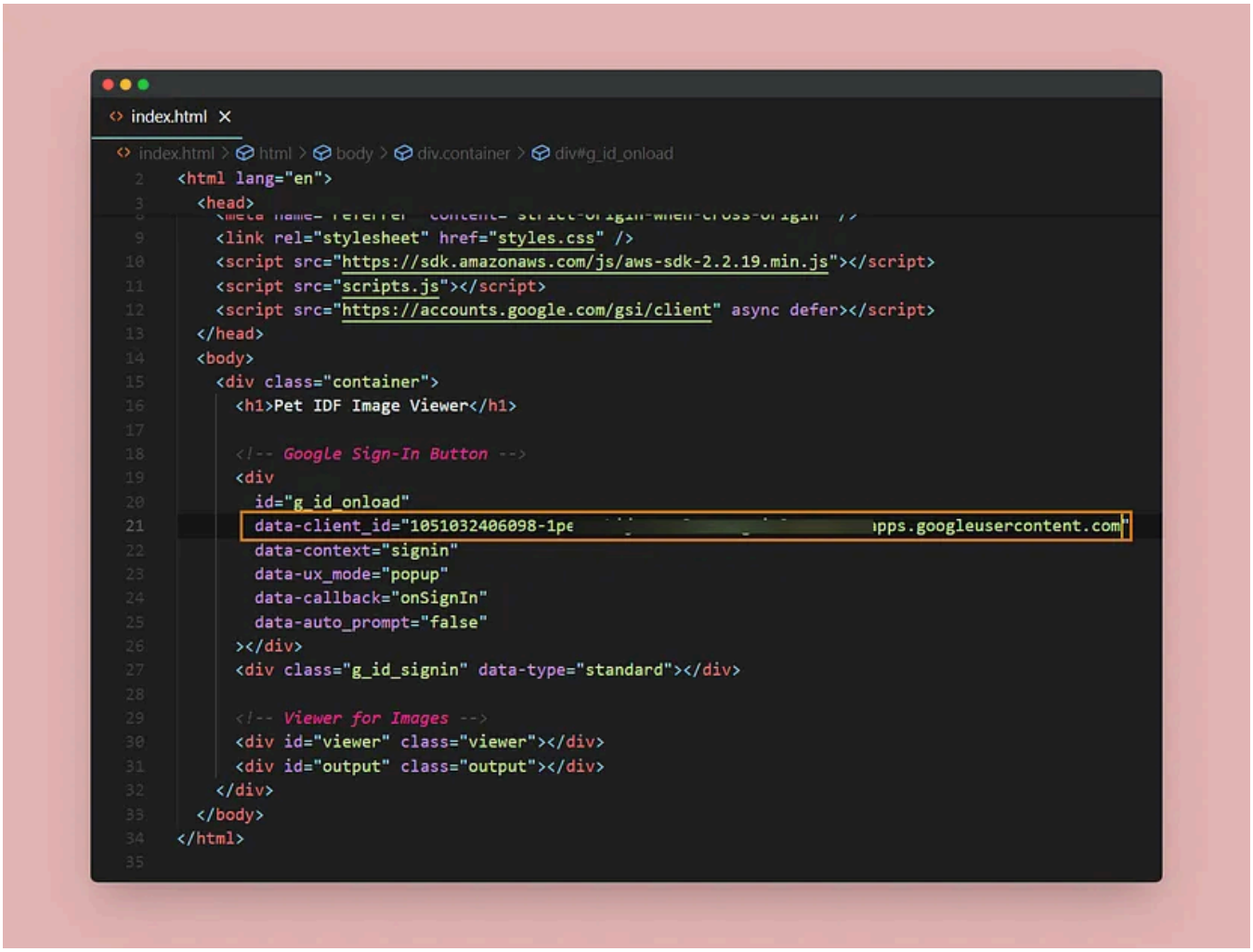
Stage 4b — Update the Files with Your Credentials

Now, open the downloaded files in a code editor to make the necessary updates.

Edit `index.html`:

- Open `index.html` in a code editor.
- Locate the placeholder `YOUR_GOOGLE_CLIENT_ID`.

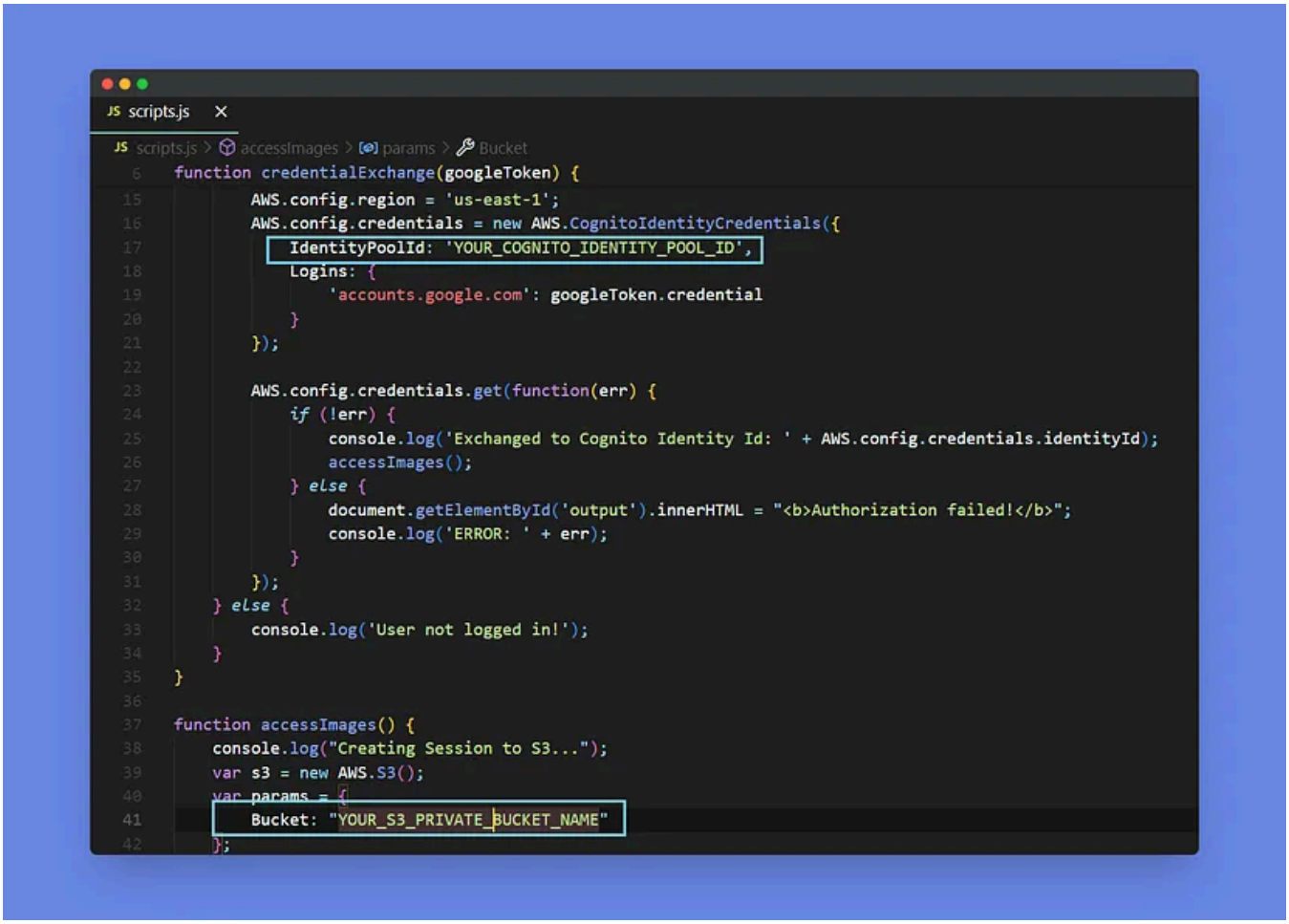
- Replace this placeholder with your **Google Client ID** from the previous setup.



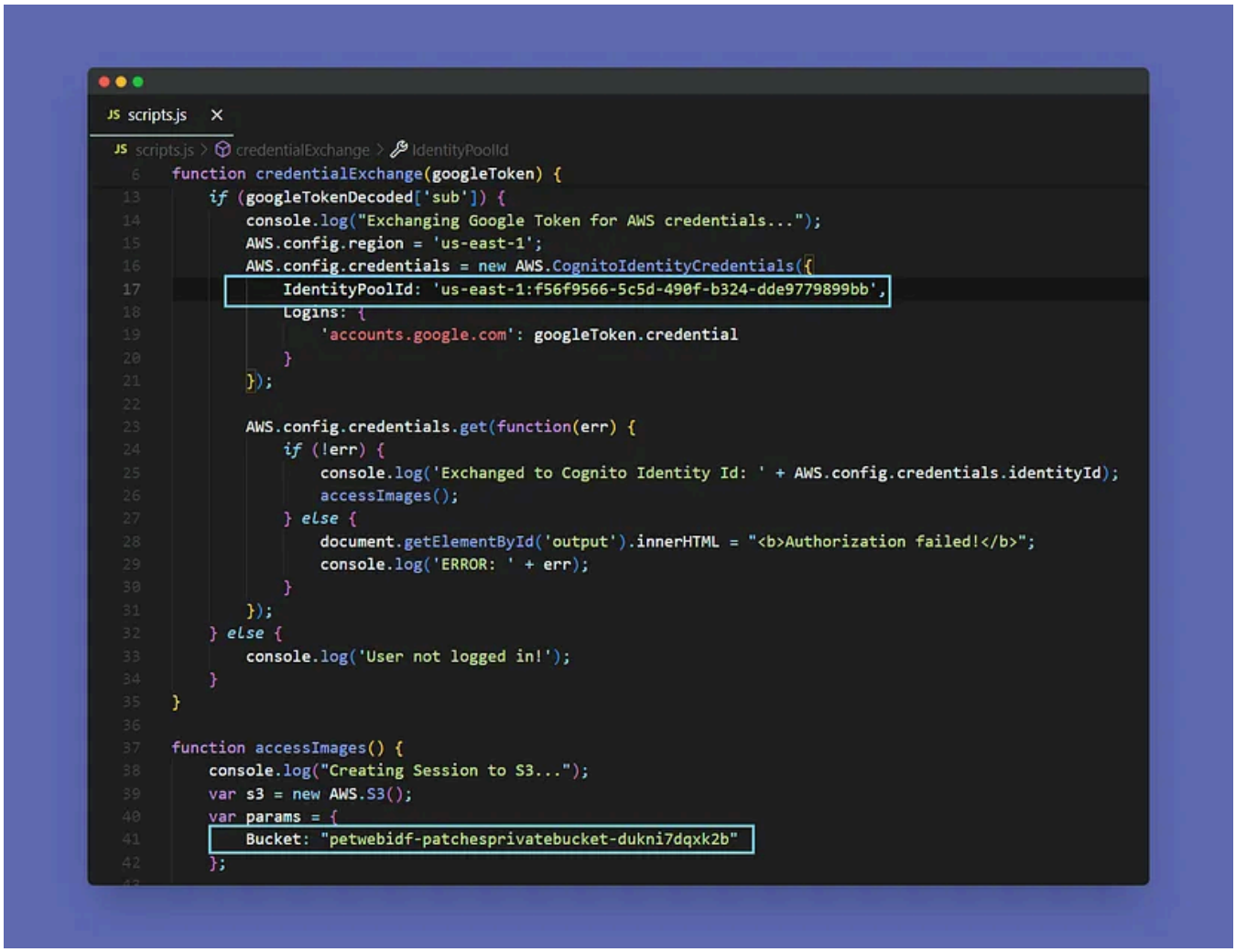
- Save the updated `index.html` file.

Edit `scripts.js`:

- Open `scripts.js` in a code editor.
- Locate `IdentityPoolId: YOUR_COGNITO_IDENTITY_POOL_ID`.
- Replace `YOUR_COGNITO_IDENTITY_POOL_ID` with your **Cognito Identity Pool ID**.



- Find `Bucket: YOUR_S3_PRIVATE_BUCKET_NAME`.
- Replace `YOUR_S3_PRIVATE_BUCKET_NAME` with the actual bucket name for **webidf-patchesprivatebucket-**.



- Save the updated `scripts.js` file.

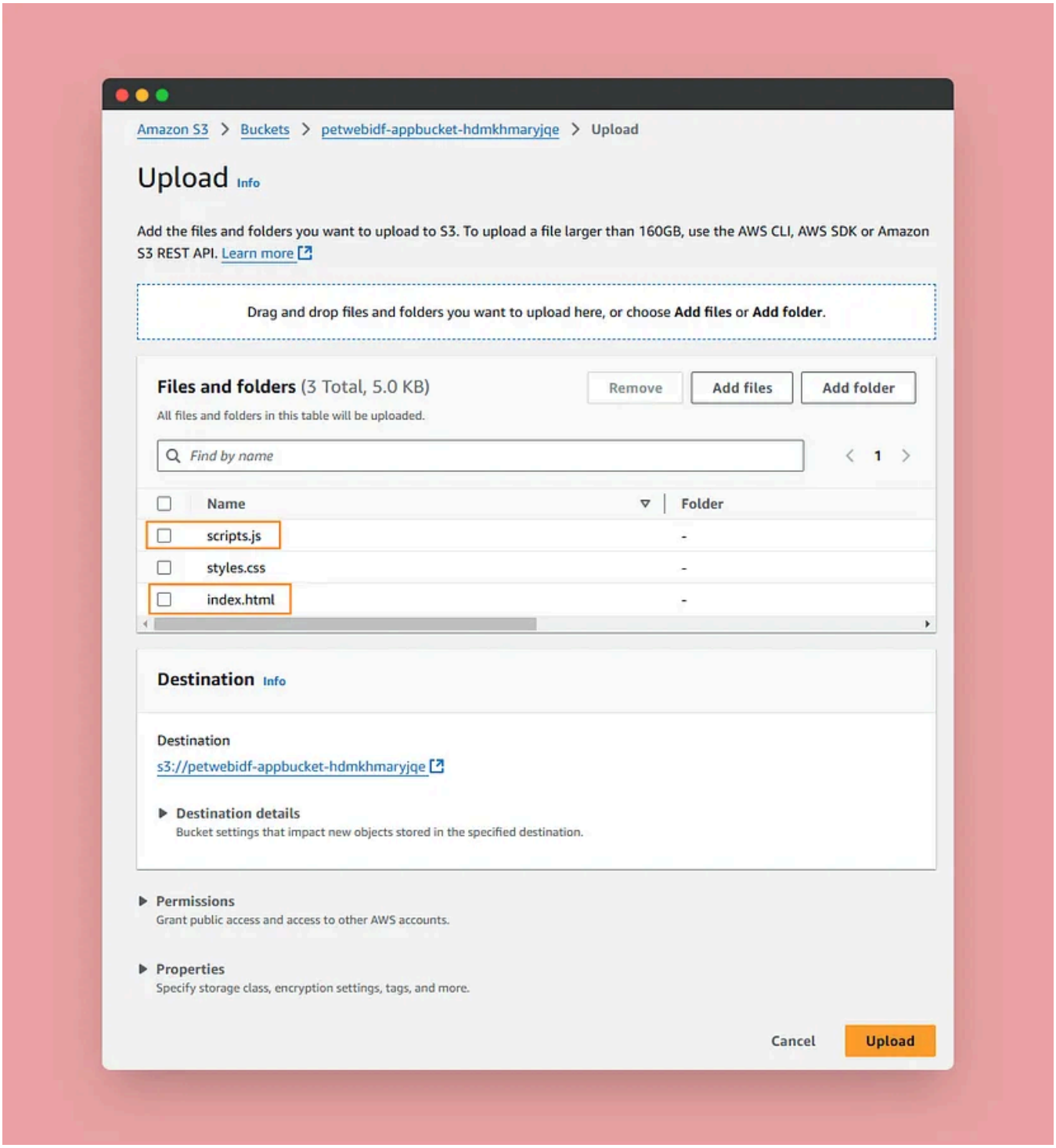
With these updates, the application can authenticate users and access the necessary AWS resources.

Stage 4c — Upload the Updated Files to S3

Now that the files are configured, upload them back to your S3 bucket.

1. **Return to the S3 Console:** Inside the **webidf-appbucket-** bucket.
2. **Upload the Files:**

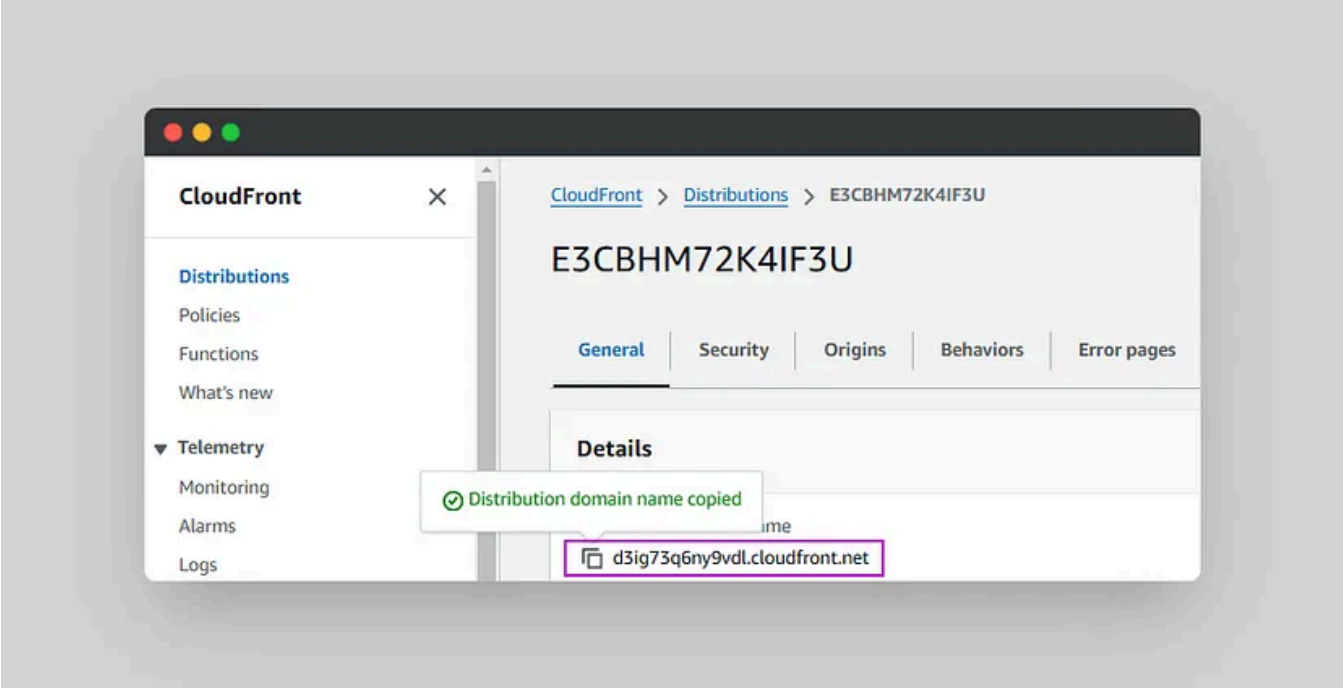
- Click **Upload.**
- Add both the updated `index.html` and `scripts.js` files.
- Click **Upload** to confirm.



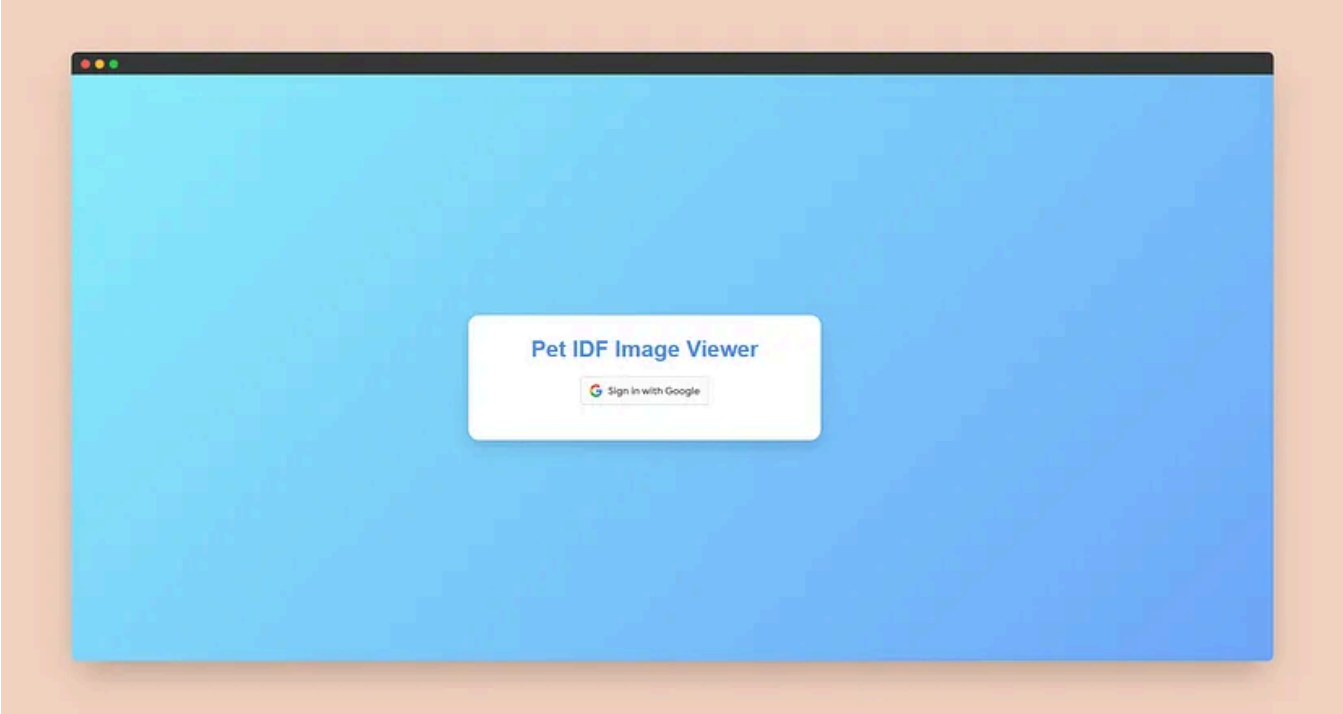
Stage 4d— Test the Application

Now, let's test the application to ensure it's configured correctly.

Open the Web Application: Open the **WebApp URL** (the CloudFront distribution URL noted in Stage 1D).



- The app should load, though it currently doesn't have access to any AWS resources.

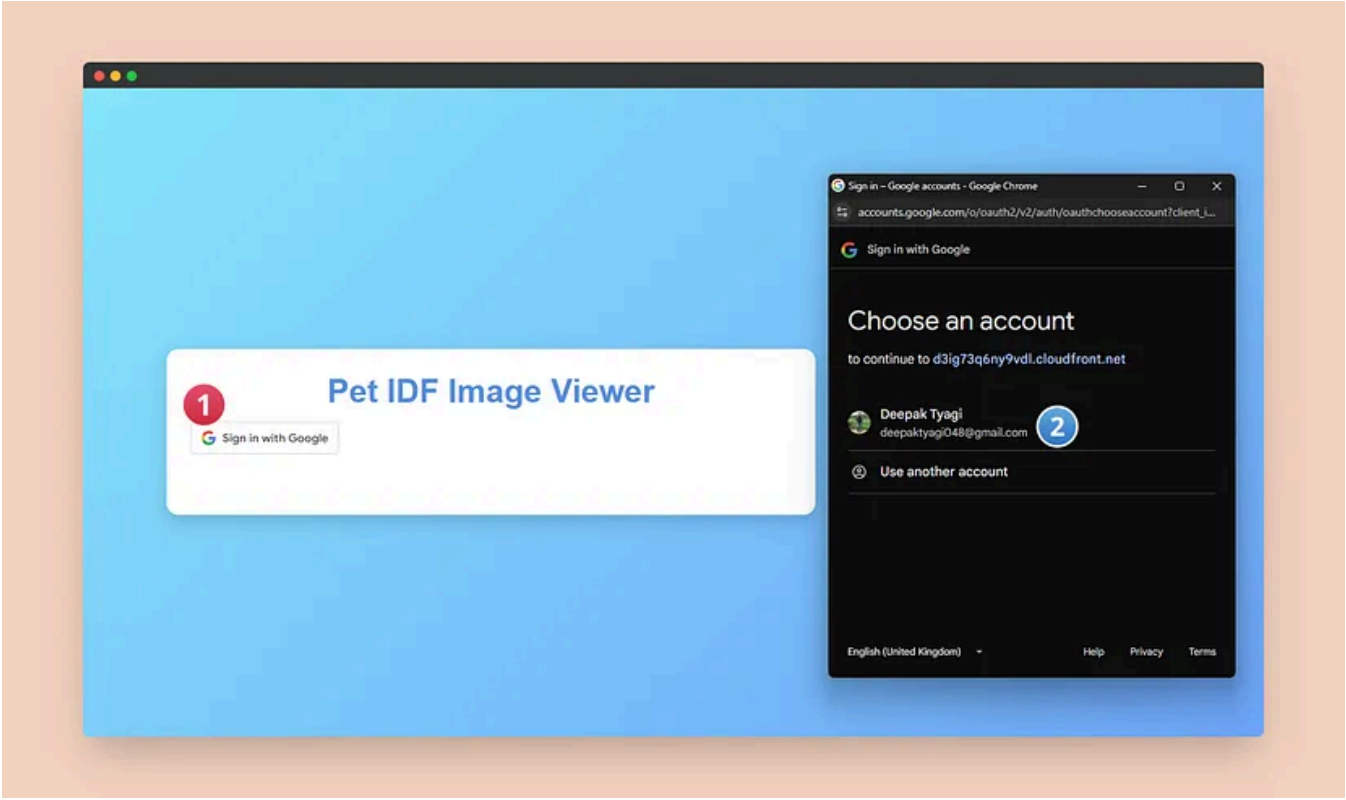


Open Browser Console:

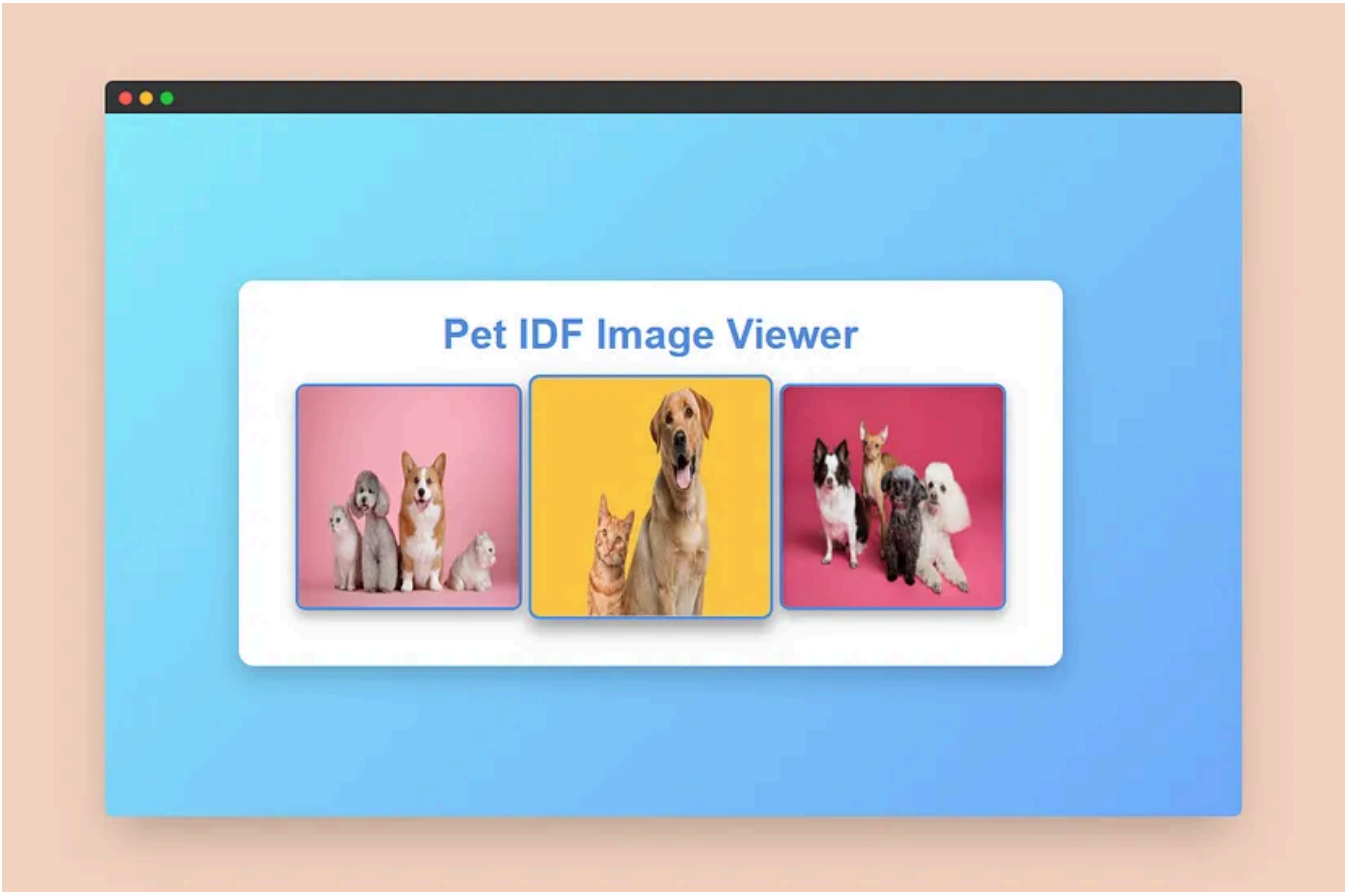
- Open your browser's web developer console (e.g., Firefox -> Browser Tools -> Web Developer Tools).
- This console will display output from the JavaScript code in the application.

Sign In:

- With the console open, click **Sign In** and sign in with your Google account.

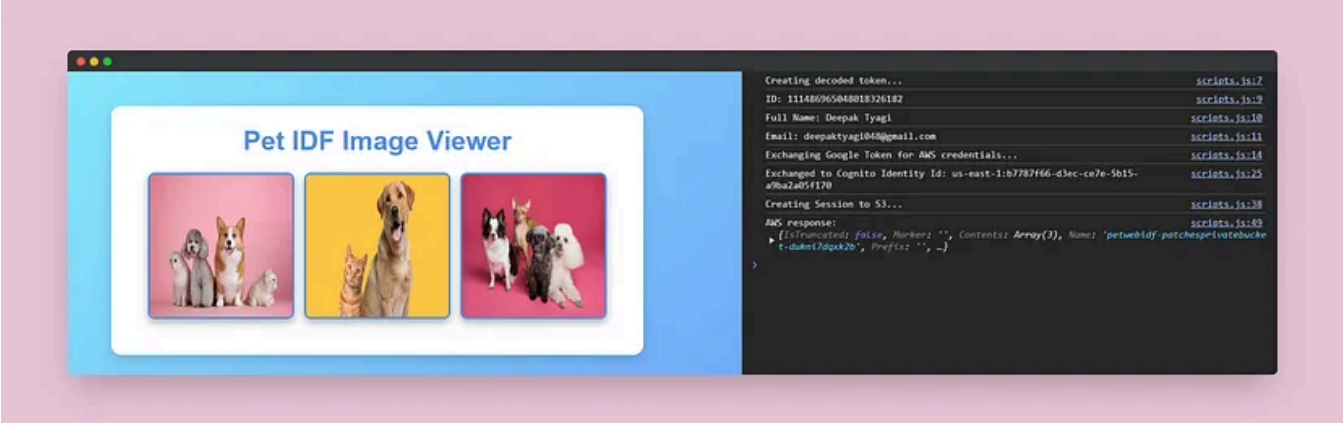


After successful sign-in, you should see the pets images populated in the viewer —

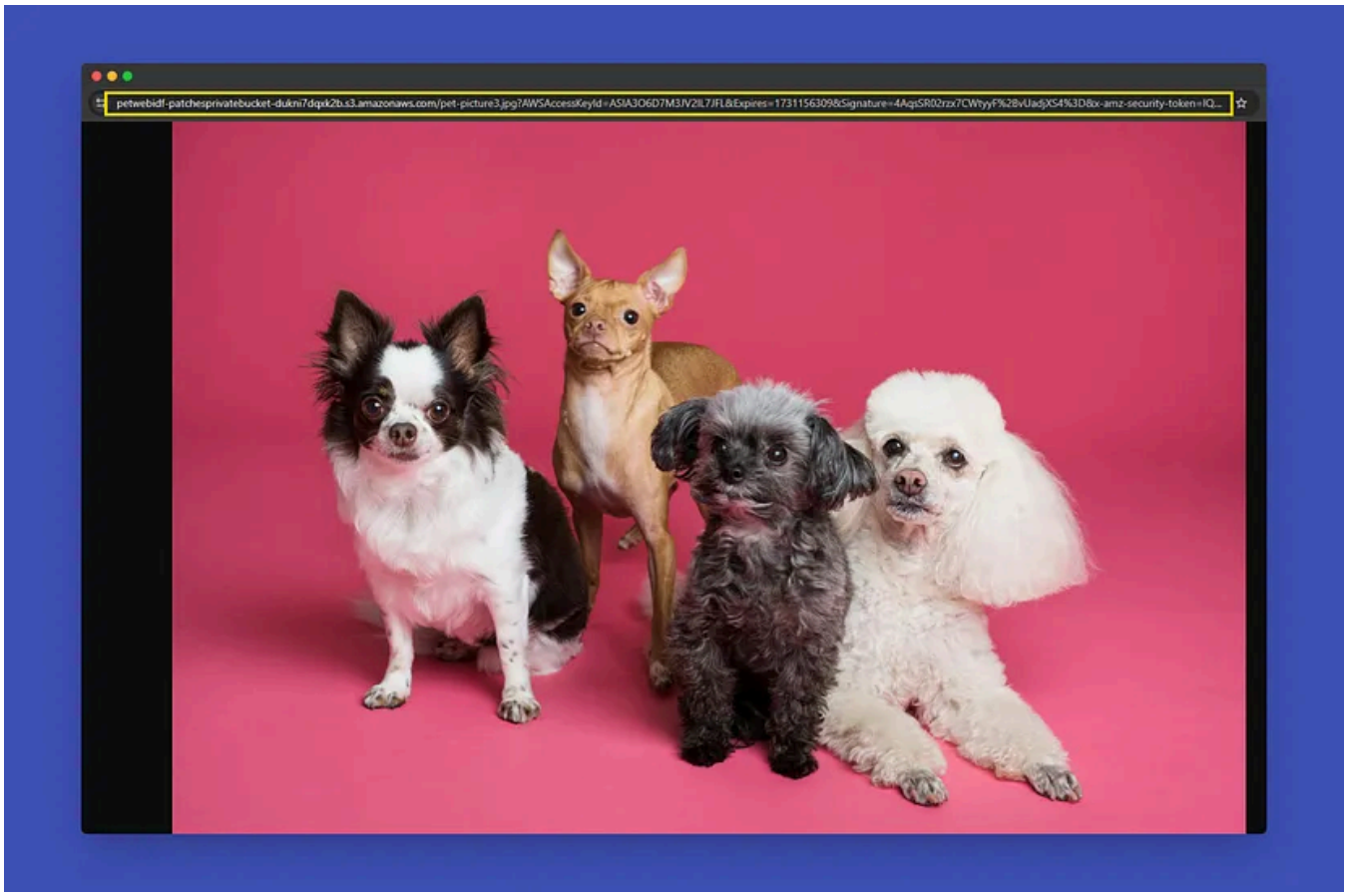


As you sign in, observe the console for the following sequence:

- **Authentication with Google IDP:** Your Google credentials authenticate with the app.
- **Access Token Retrieval:** Google returns an access token.
- **Cognito Token Exchange:** The token is exchanged with AWS Cognito, generating temporary AWS credentials.
- **S3 Access:** Using these credentials, the app lists and retrieves objects from the private S3 bucket.
- **Image Display:** Presigned URLs are generated for each image in the private bucket, allowing them to load in the browser. You should see three cat pictures displayed.



Click Each Image: Notice the URLs used — each is a presigned URL generated by the JavaScript code in your browser, using the Cognito-generated credentials.



This setup is fully serverless, with no need to manage any compute resources yourself.

Handling Caching Issues

If you encounter issues with signing in, check the following:

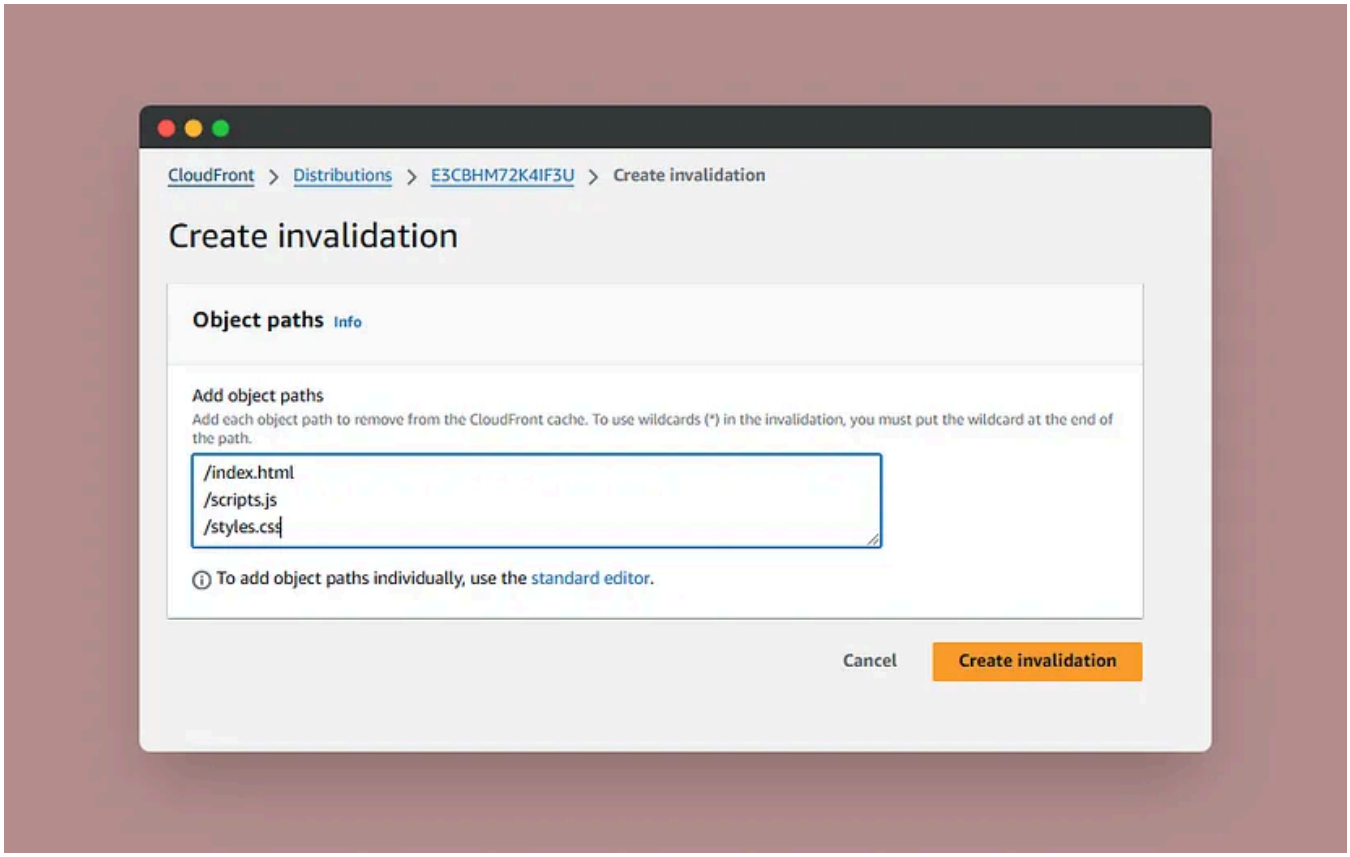
- Ensure that the **index.html** and **scripts.js** files are uploaded with the correct credentials.
- Verify that your **CloudFront distribution** has finished deploying the latest files.

If issues persist, the CloudFront distribution may be caching older versions of the files. To resolve this:

1. **Go to the CloudFront Console.**
2. **Create an Invalidation:**

- Select the relevant CloudFront distribution.
- Click on the **Invalidations** tab.
- Choose **Create invalidation**.

- Enter `/index.html` , `/styles.css` and `/scripts.js` in the **Object Paths** field.
- Click **Create invalidation** to clear the cache.



Stage 4 — Finish

At this stage, you have a fully functional serverless application with the following components:

- **Front-End App Bucket:** Holds the application files.
- **Configured Google API Project:** Enables Google-based authentication.
- **Cognito Identity Pool:** Manages user identities and authentication.
- **IAM Roles for the Identity Pool:** Grants necessary permissions.
- **HTML and JavaScript Configured:** Connects to the Google IDP and Cognito for authentication and S3 access.

This setup provides a secure, serverless application that leverages AWS resources efficiently. Congratulations on building a fully operational, serverless web application!

Step5 — Resource Cleanup

Once you're done using your **PetIDF Serverless Web Application** or if you want to avoid any ongoing charges, it's essential to properly delete the resources that were set up during the deployment stages.

In this final stage, we'll walk through how to delete each component, including the **Google API project**, **Amazon Cognito identity pool**, **IAM roles**, and **CloudFormation stack.**

Stage 5a — Delete the Google API Project & Credentials

Go to the **Google Cloud Resource Manager**:

- Navigate to Google Cloud Console.
- Locate and select the project named **PetIDF**.

Delete the Project:

- Click **DELETE.**
- Enter the project ID (which may be slightly different from the name shown).
- Confirm by clicking **Shut Down.**

Stage 5b — Delete the Cognito Identity Pool

Open the **AWS Cognito Console**:

- Go to Cognito Console.
- Click **Federated Identities.**

Delete the Identity Pool:

- Select the **WebIDF** identity pool.
- Click **Edit Identity Pool.**
- Expand the **Delete identity pool** section.
- Confirm by clicking **Delete Identity Pool** and then **Delete Pool.**

Stage 5c — Delete the IAM Roles

Open the **AWS IAM Console**:

- Navigate to IAM Console.

Delete **Cognito Roles**:

- In the **Roles** section, find the roles starting with **CognitoRole_PetIDF**.
- Select both roles, click **Delete Role**, and confirm by clicking **Yes, Delete**.

Stage 5d — Delete the CloudFormation Stack

Open the **AWS CloudFormation Console**:

- Go to CloudFormation Console.

Delete the Stack:

- Find the **PetWebIDF** stack, select it, and click **Delete Stack**.
- Confirm by clicking **Delete Stack**.

Final Verification

After completing these steps, all associated resources should be successfully deleted, including:

- Google API Project and OAuth credentials
- Cognito Identity Pool
- IAM Roles for the Identity Pool
- CloudFormation stack and associated AWS resources

By following this process, you ensure that no lingering resources incur charges and maintain a clean AWS environment.

Conclusion

Well done! You've successfully set up, configured, and safely torn down a complete **PetIDF Serverless Web Application** using AWS and Google Auth. Through this project, you've deployed serverless infrastructure, configured secure user

authentication with **Google OAuth** and **AWS Cognito**, and leveraged **CloudFormation** for efficient resource management. By following best practices in security and cleanup, you've created a scalable, cost-effective application foundation.

Thank you for following along — this project marks a great step into the world of cloud-native, serverless solutions!

If you've discovered any value or gained insights from this blog, I would greatly appreciate it if you could show your appreciation by giving this story a clap.

Please feel free to reach out to me on LinkedIn with any questions, or you can drop them into the comment section below. I would be more than happy to assist you.

AWS | Serverless | Security | DevOps | Cloud Computing

Follow

Written by Deepak Tyagi

48 Followers · 4 Following

DevOps & Cloud Enthusiast | AWS | Docker | Jenkins | Terraform | Git | Kubernetes | Linux

No responses yet

What are your thoughts?

Respond

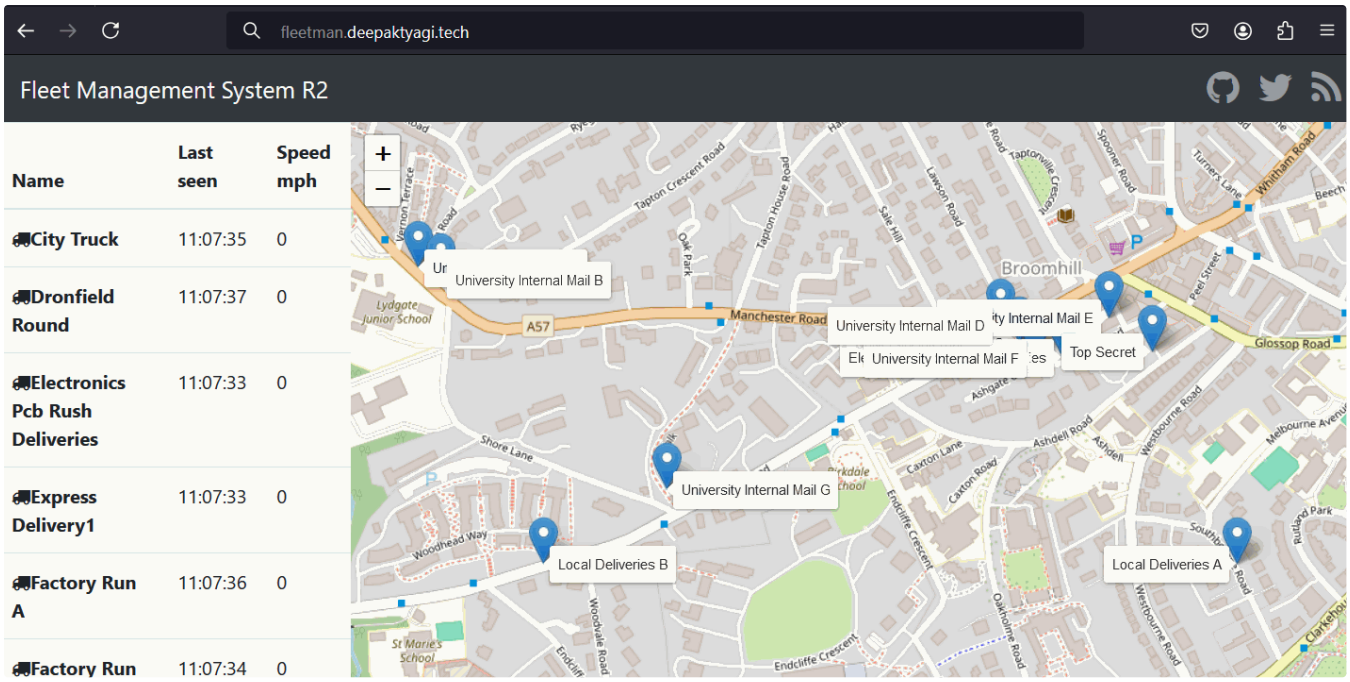
More from Deepak Tyagi

Deepak Tyagi

How I passed the AWS Solutions Architect Associate Exam ?

Table of Contents

Oct 9, 2023 62

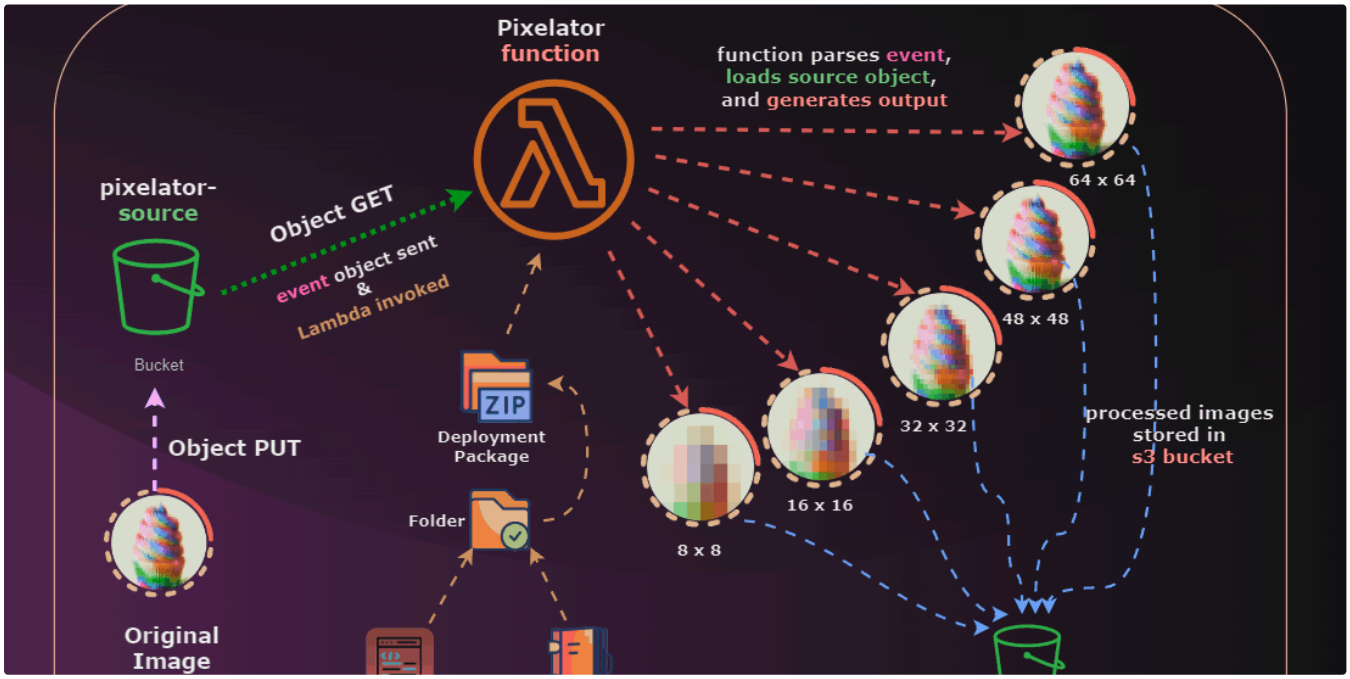


Deepak Tyagi

Mastering Production-Grade Microservices: A Robust 6-Tier EKS Deployment with Jenkins, Terraform...

In today's fast-paced DevOps world, deploying microservices at scale demands a robust and integrated approach. This guide dives into a...

Aug 30, 2024 65 1



Deepak Tyagi

Image Pixelator: Event-Driven Real-Time S3-to-Lambda Image Processing Workflow using Terraform

In this blog, we'll explore how to build an event-driven image processing pipeline using AWS Lambda, S3, and Terraform. When an image is...

Oct 13, 2024 41

Deepak Tyagi

The Nostalgia: Deploying Snake Xenzia Game on S3 using Terraform

Remember the good old days when mobile phones were just phones, and the most exciting thing they could do was play Snake? Ah, the...

Oct 18, 2023 32

See all from Deepak Tyagi

Recommended from Medium

Jonas Neumann

This New AWS Serverless Stack could be a Game-Changer

Going Serverless without vendor lock-in is now possible

Jan 24 160 5

howtouselinux

Grafana Got Better: Smarter Dashboards, Easier Sharing & More in the New Release!

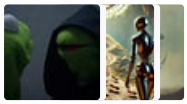
What is Grafana? 🤔

Feb 4 29

Lists

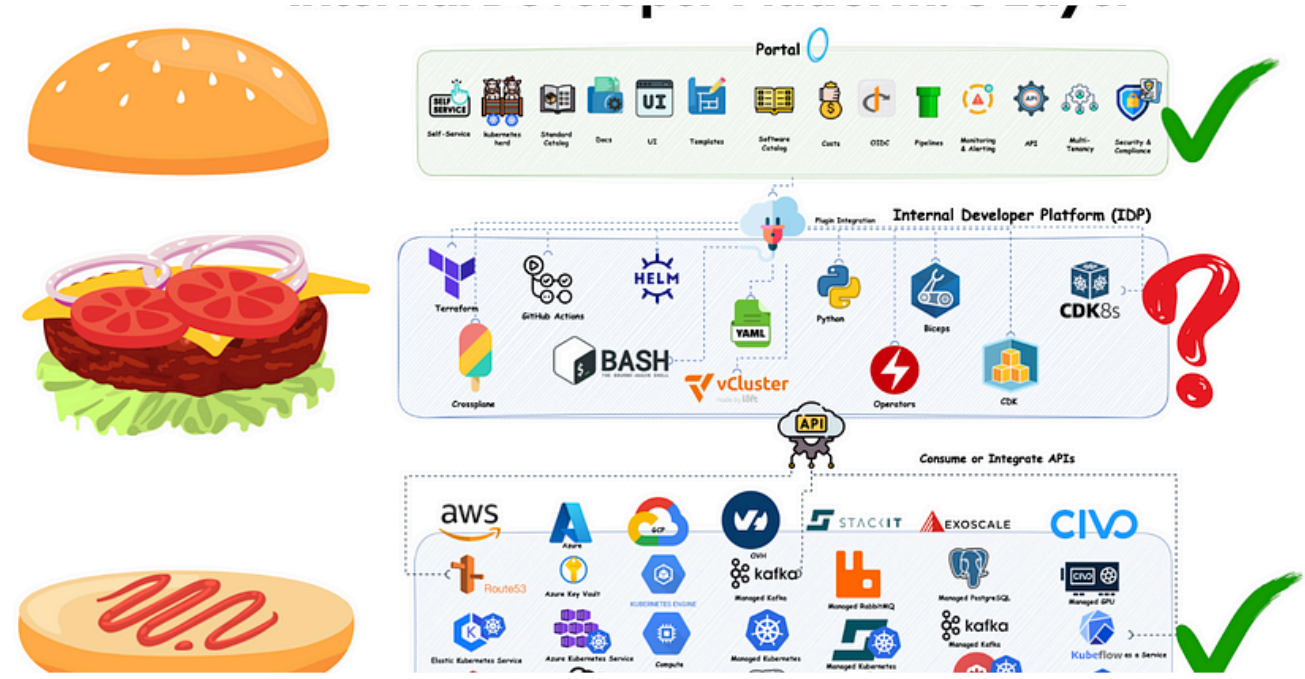
General Coding Knowledge
20 stories · 1906 saves

Leadership
62 stories · 515 saves



Natural Language Processing
1928 stories · 1582 saves

Productivity
244 stories · 674 saves

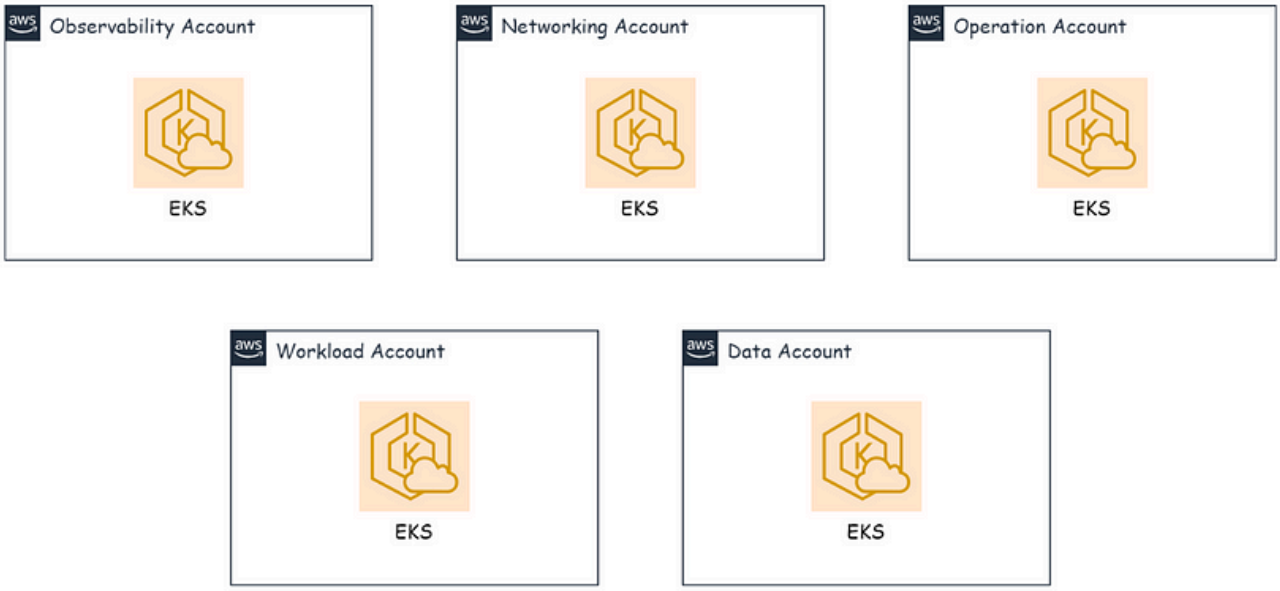


In ITNEXT by Artem Lajko

How to Build a Multi-Tenancy Internal Developer Platform with GitOps and vCluster

The key role of Platform Engineering teams in building a Kubernetes based Platforms.

5d ago 169 2

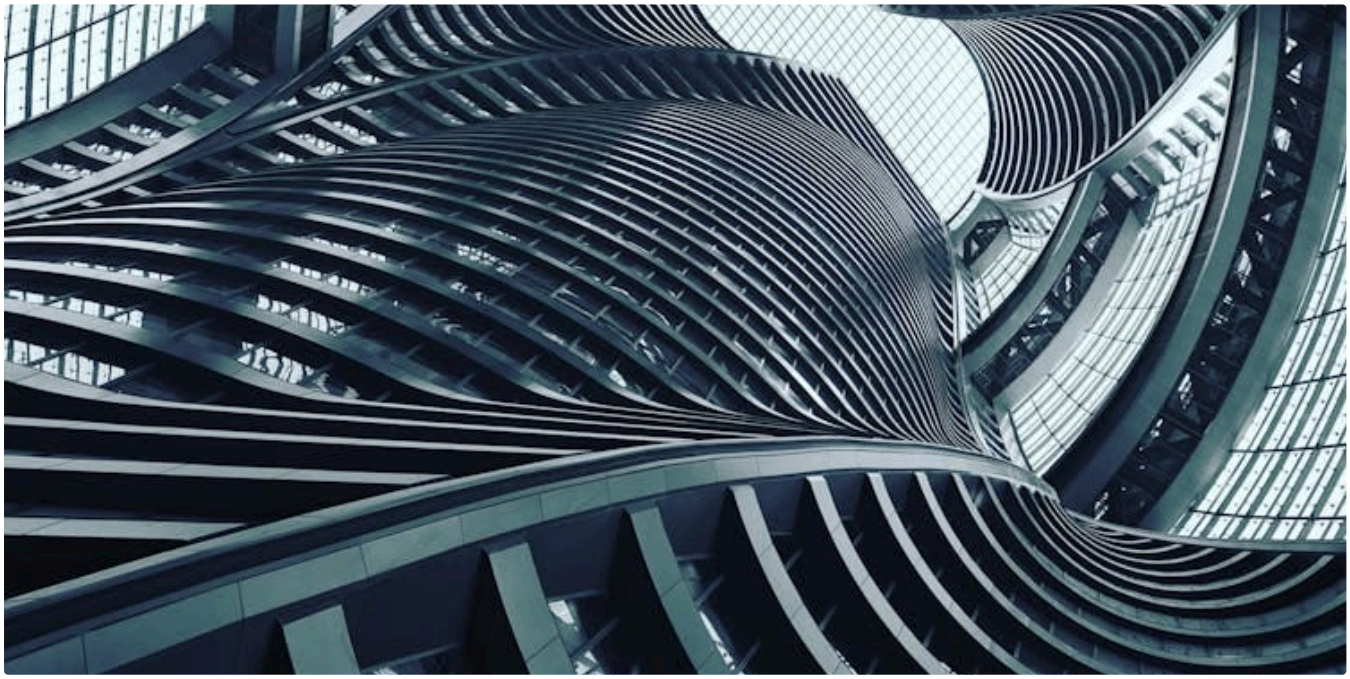


In AWS in Plain English by Quân Huỳnh

Kubernetes for Multi-AWS Account Strategy: Infrastructure for Scale

The Banking Infrastructure on Cloud series shares insights about banking systems' infrastructure architecture on the AWS Cloud.

Feb 2 47



Emmanuel Meric de Bellefon

Why you shouldn't use NestJS

Being in the NodeJS ecosystem for a few years, I'm surprised by how many startups use NestJS and still face the same issues. Probably a lot...

Feb 1 231 44

In AWS Tip by Tobias Blaser

Mastering Cross-Account Communication in AWS Serverless Architecture

A comprehensive guide to implementing secure and scalable communication patterns across AWS accounts for serverless applications

Jan 28

See more recommendations