Quick tip: Review the prerequisites before you run the lab

End Lab 00:53:54

Caution: When you are in the console, do not deviate from the lab instructions. Doing so may cause your account to be blocked. Learn more.

Open Google Cloud Console

GCP Username

student-00-f72d743b57a4

GCP Password

HdnHYbTa3zDs

GCP Project

qwiklabs-gcp-00-99c1f98

Deploy and Secure a GenAI Web Application

Lab 2 hours No cost Intermediate

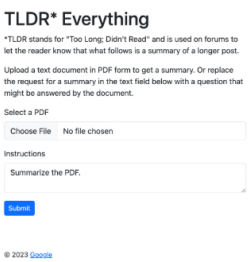
★★★★☆ Rate Lab

This lab may incorporate AI tools to support your learning.

Previous Next

Overview

In this lab, you will deploy a minimal GenAI web app with a UI so it can be used by non-technical team members. The app you'll deploy allows a user to upload a text PDF and request a summary or ask a question that the language model can answer using the



The architecture you'll use here could be repurposed to share many AI tools you could create, for example:

- internal tools to accelerate or enhance the productivity of teams in your organization
- proof-of-concepts for potential customers
- GenAI utilities to speed up your own workflows

What you will learn

- How to deploy a web app with an attractive UI, but with minimal effort put into styling the front end.
- How to use Cloud Run's deployment options to deploy with a CICD pipeline that will update your Cloud Run service on updates to a code repository branch.
- How to use Identity-Aware Proxy to limit access to a web app to only authorized users. Remember, Generative AI models can be expensive to query, so we don't want bad actors online to be able to find our app and make queries that cost us a lot or expose sensitive data.

Setup and requirements

Read these instructions. Labs are timed and you cannot pause them. The timer, which starts when you click **Start Lab**, shows how long Google Cloud resources will be made available to you.

This Qwiklabs hands-on lab lets you do the lab activities yourself in a real cloud environment, not in a simulation or demo environment. It does so by giving you new, temporary credentials that you use to sign in and access Google Cloud for the duration of the lab.

What you need

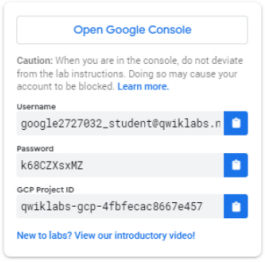
To complete this lab, you need:

- Access to a standard internet browser (Chrome browser recommended).
- Time to complete the lab.

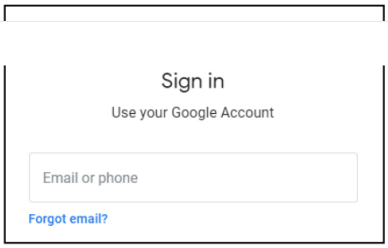
Note: If you already have your own personal Google Cloud account or project, do not use it for this lab.

Note: If you are using a Pixelbook, open an Incognito window to run this lab.

1. Click the **Start Lab** button. If you need to pay for the lab, a pop-up opens for you to select your payment method. On the left is a panel populated with the temporary credentials that you must use for this lab.

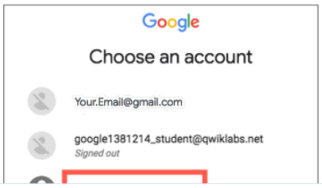


2. Copy the username, and then click **Open Google Console**. The lab spins up resources, and then opens another tab that shows the **Sign in** page.



Tip: Open the tabs in separate windows, side-by-side.

If you see the **Choose an account** page, click **Use Another Account**.



3. In the **Sign in** page, paste the username that you copied from the Connection Details panel. Then copy and paste the password.

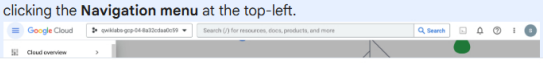
Important: You must use the credentials from the Connection Details panel. Do not use your Qwiklabs credentials. If you have your own Google Cloud account, do not use it for this lab (avoids incurring charges).

4. Click through the subsequent pages:

- Accept the terms and conditions.
- Do not add recovery options or two-factor authentication (because this is a temporary account).
- Do not sign up for free trials.

After a few moments, the Cloud Console opens in this tab.

Note: You can view the menu with a list of Google Cloud Products and Services by clicking the **Navigation menu** at the top-left.



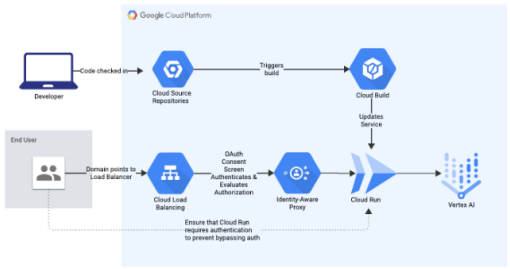
Task 1. Understand our app

Task 1. Understand our app & architecture diagram

This app is written in Python using the popular web microframework Flask. It uses a package called Bootstrap Flask to easily implement the Bootstrap front-end framework in a Flask app. Bootstrap makes some default design decisions for us so you can present a nice UI without needing to design it from scratch yourself.

Below is an architecture diagram of what you'll deploy. Consider the top half first, then we'll look at the bottom half.

You'd like to make it easy to maintain and improve your app, so you will use a deployment option that Cloud Run provides to deploy the app using a Cloud Build Trigger. This allows for developers to update the code base by testing locally and then version of the service to Cloud Run.



The bottom half of the diagram shows how you can protect your app from unauthorized access using Identity-Aware Proxy. Identity-Aware Proxy allows you to control access to a resource without needing to implement a sign-in screen, a forgot password flow, and other elements of a user management system. Instead you can grant access to users, Google Groups, or whole domains by assigning them the IAM role of "IAP-secured Web App User" using the Google Cloud Console.

As a whole, this pattern allows for quick deployment & maintenance of a simple web app designed to serve internal users or a small pool of external users.

Task 2. Push the code to a Github repository & review it

1. Open Cloud Shell by selecting the **Cloud Shell** icon at the top of the Google Cloud Console ( ) and selecting **Continue**.

2. Download the app's code using a `gcloud storage cp` command with the recursive flag(`-r`) to download a code directory and its contents. This step stands in for you developing the app locally:

```
gcloud storage cp -r gs://pls-resource-bucket/tldr-
```

3. If prompted to authorize Cloud Shell, select **Authorize**.

4. This folder of code isn't a git repository yet, so let's navigate into the folder and initialize the directory as a repo.

```
cd tldr-everything
git init
git checkout -b main
git add .
git config --global user.email "you@example.com"
git config --global user.name "Qwiklabs Demo Account"
git commit -m "initial commit"
```

5. Open a new browser tab, navigate to GitHub.com and sign in. If you don't have a Github account, you can create one for free. You'll return to this tab to create a repo shortly.

6. Back in your GCP Console tab, open up Cloud Shell, authenticate with GitHub using the command line:

```
-
```

7. Select **GitHub.com** as the type of account to log into, then **HTTPS** for the protocol, confirm you will authenticate Git with GitHub credentials with a **Y**, and

then select **Login with a web browser**.

8. Copy the one-time code provided to you by the `gh` command line tool. Then press **Enter** and click the link to go to `https://github.com/login/device`. Login. You may need to complete additional steps to login and confirm your account if you have multi-factor authentication enabled (as you should).

9. You will then be prompted to paste the code displayed in Cloud Shell after the `gh auth login` step above. Paste the code and proceed.

10. Select **Authorize GitHub**. You should see a message like "Congratulations, you're all set!" You can close this tab.

11. In the tab where you originally logged in to GitHub, select the **New** button () to create a new repo.

12. Give your repo the name **summarizer-app**.

14. Then select **Create repository**.

15. In your new repository, use the copy button to copy the commands under the header **...or push an existing repository from the command line**. They should look like this, but be specific to your account:

```
git remote add origin https://github.com/YOUR-ACCOUNT-
NAME/summarizer-app.git
git branch -M main
git push -u origin main
```

> **Note:** If GitHub displays a 'Permission denied (publickey)' error, generate an SSH key using Cloud Shell and add it to your GitHub account to grant access to Git. You can follow the steps outlined in the GitHub documentation."

16. Return to Cloud Shell to paste this code copied from GitHub, and run it. After the code is pushed, you can close the Cloud Shell pane.

`main` branch is selected, with code files listed, and a README on the page.

18. Select `app.py` to open our app's main file and scroll to the import statements.

```
import logging
import vertexai
from vertexai.generative_models import GenerativeModel,
GenerationConfig

from langchain.document_loaders import PyPDFLoader  # to
load and parse PDFs
import markdown  # to format LLM output for web display

import os  # to remove temp PDF files
import uuid  # to generate temporary PDF filenames

from flask import Flask, render_template, redirect,
url_for, session
from flask_bootstrap import Bootstrap5
from flask_wtf import FlaskForm, CSRFProtect
from flask_wtf.file import FileField, FileRequired,
FileAllowed
from wtforms.fields import SubmitField, TextAreaField
```

19. Notice a few key packages you are importing:

package	note
`from langchain.document_loaders import PyPDFLoader`	langchain gives you a few convenient Document loader classes to import and work with different sources of text.
`wtforms` and `flask_wtf`	WTForms lets you build forms in your Flask app using Python.
`markdown`	This package can help format text, like that from your LLM output, for web display.

20. Below the imports and a few configuration & initialization lines, notice the class definition for our upload form written in Python. See that it includes file-type validation from the `wtforms` and `flask_wtf packages`. This upload field will accept only files with `.pdf` extensions:

```
class UploadForm(FlaskForm):
    """A basic form to upload a file and take a text
```

```
        validators=[FileRequired(),
```

```
FileAllowed(['pdf'], 'Please select a PDF.')],
        label="Select a PDF",
    )
    text_input = TextAreaField(label="Instructions",
default="Summarize the PDF.")
    submit = SubmitField()
```

21. Below this class, we see the intialization of our Vertex AI client and the configuration of our model: For this use case, we are selecting a low temperature for more accuracy (rather than variation) in our model's response.

    Working with top_p means that for each word the model generates, we are restricting its choice of words to the minimal set of words that will account for this portion of the probability distribution of possible next words. So a smaller top_p means choosing from a more restricted and predictable word each time.

```
vertexai.init(project=os.environ["PROJECT_ID"],
location="us-central1")
```

```
generation_config = GenerationConfig(
    temperature=0.3,
    top_p=0.6,
    candidate_count=1,
    max_output_tokens=4096,
)
```

22. At our default Flask route, the application displays the form by instantiating the `UploadForm()` class created above. When the user submits the form, the app checks if the validations have passed, then creates a temporary filename to save the PDF, load its text using langchain's `PyPDFLoader`, and combine the multiple pages of content into `combined_text`.

```
@app.route("/", methods=["GET", "POST"])
def index():
    """Route to display the input form and query the
LLM."""
    form = UploadForm()
    if form.validate_on_submit():
        pdf_temp_filename = str(uuid.uuid4())
        form.pdf_file.data.save(pdf_temp_filename)
```

```
in pages])
```

23. One way we could improve this app is to implement a chain to handle longer (or multiple) documents, but for now, we will use the general guidance that 100 tokens correspond to about 60-80 words, giving us a conservative ratio of about 1.66 tokens per word (100 / 60 = 1.667) to only call our model on texts we believe will be short enough for us to process using the model's token limit.

    You can see here our prompt is very simple, opening with the input to our UI's 'Instructions' text field, then specifying that the model should use information from the PDF to respond, then a label to identify the PDF text ("PDF:") and the PDF text itself.

```
        word_count = len(combined_text.split())
        # 18500 words times ~1.66 tokens per words should
keep us under Gemini's token limit:
        # https://ai.google.dev/models/gemini#model-
variations
        if word_count < 18500:
            prompt = f"Instructions: {form.text_input.data}
```

```
            logging.debug(prompt)
            response = model.generate_content(prompt,
generation_config=generation_config)
            response_text = response.text.replace("•", "
*")
        else:
            response_text = "This text is too long for this
application's current configuration.\n\nPlease use a
shorter text."
```

24. Finally, we parse the response using the `markdown` package to convert the model's response to HTML to display it on our pdf_results template:

```
        markdown_response =
markdown.markdown(response_text)
        session["markdown_response"] = markdown_response
        logging.debug(f"Response: \n{markdown_response}")
        os.remove(pdf_temp_filename)
        return redirect(url_for("pdf_results"))
```

on the port 8080, which is the port Cloud Run sends requests to by default.

26. While we haven't gone over all the code in the app, the above highlights cover our key elements.

Task 3. Enable the Vertex AI APIs

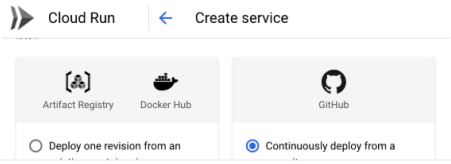
1. Navigate to Vertex AI and then click **Enable All Recommended APIs**.

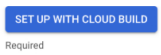
Click *Check my progress* to verify the objective.

Task 4. Deploy a Cloud Run Service

1. Search for or navigate to Cloud Run in the Cloud Console and select "Service" under **Deploy Container**.

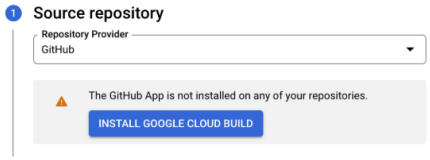
2. Select the radio button for "**Continously deploy from a repository**". This option will deploy a Cloud Build Trigger so that an update to a given branch in our repo will deploy updates to our app. Select the "**Set up with Cloud Build**" button.





3. Select the "GitHub" Repository Provider and the **Authenticate** link in the tool tip below it.

4. If prompted, select the **Install Google Cloud Build** button.



5. If prompted for a Google identity, select your Qwiklab student credentials to proceed.

6. On the GitHub "Install Google Cloud Build" page, select your GitHub username. You may choose to install Google Cloud Build on all repositories. If in doubt (`summarizer-app`), select **Install**.

7. Back in the Google Cloud Console, in the "Set up with Cloud Build" panel, select the "summarizer-app" Repository. Confirm the terms by **selecting the checkbox**, then select **Next**.

8. In the Buld Configuration settings, keep the Branch set to `^main$`, so only pushes or merges to your `main` branch will trigger deployments. In a real deployment setting, you could set up separate Cloud Run services, perhaps in different environment projects, to deploy from `dev`, `test`, and `main` branches.

9. For Build Type select "Dockerfile" and leave the Source location to the default of "/Dockerfile" because the Dockerfile for this deployment is at the root level of this app's repo structure. Click "Save".

10. Back in the Cloud Run deployment settings, ensure the Region is set to `us-west1`.

Note: Cloud Run displays only the region where the lab is provisioned.

authentication" after configuring Identity-Aware Proxy.

12. Under "Billing", select **Request-based** (Charged only when processing requests. CPU is limited outside of requests.)

13. Under "Service autoscaling", set "Minimum number of instances" to 0. You could scale this up if you did not want users to have to wait for instances to spin up when no one has used the app recently.

14. Expand the "**Container(s), Volumes, Networking, Security**" section by selecting its name, and select the **Containers** tab.

15. Within the Containers tab, scroll to "Revision autoscaling" and set the **Maximum number of instances** to 1.

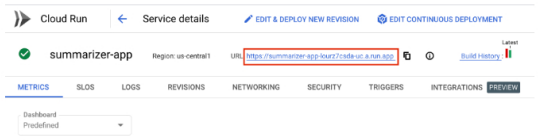
16. Scroll back up to see tabs for "Settings", "Variables & Secrets" and "Volume Mounts" within the Containers tab, and select the **Variables & Secrets** tab.

17. Under "Environment variables", select **+ Add Variable**. Create a variable with a Name of "PROJECT_ID" and the value of `qwiklabs-gcp-00-99c1f980927b`.

19. Navigate to Cloud Build in the Google Cloud Console, then click the "Triggers" navigation item on the left to see the Trigger that has been created for this service.

20. You can also select the "History" nav item on the left to see the current Build. Select its "Build" hash value to see the steps in progress.

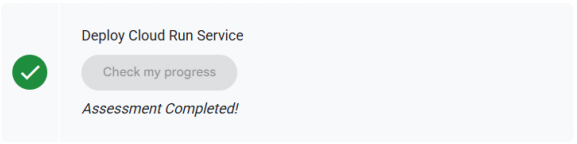
21. When the build has completed, navigate back to Cloud Run and select the service. Select the service's URL to access it:



22. Try uploading a PDF (if you need one as an example, here is a report on 2023 Google Carbon Removal Research Awards) and viewing the summary you receive.

23. Try uploading a PDF and replacing the instructions to summarize the content with removal?").

Click *Check my progress* to verify the objective.



Task 5. Put the app behind a Load Balancer

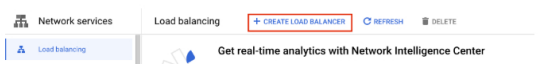
To protect your Cloud Run service with Identity-Aware Proxy, first put it behind a load balancer.

2. From the IP Addresses page, select **Reserve external static IP address**, name the IP something like "summarizer-app-lb-ip", set the Type to "Global" and select "Reserve".

3. Copy and paste the IP Address assigned to you in a notes document to use a few steps later.

4. Navigate to "Load balancing" by searching for it at the top of the Cloud Console and selecting the option associated with "Network services".

5. Select **Create Load Balancer**



6. Select **Application Load Balancer (HTTP/HTTPS)** and **Next**.

7. Select **Public facing (external)** and **Next**.

9. Select **Global external Application Load Balancer** and **Next**.

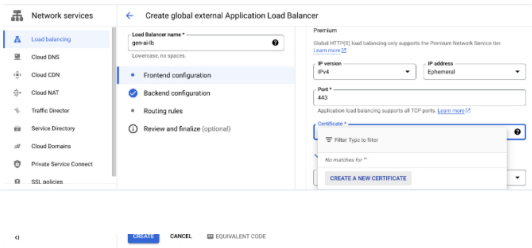
10. Select **Configure**.

11. Name the load balancer (ex. "summarizer-app-lb"), then on the right-hand side, name the frontend (ex. "summarizer-app-frontend").

12. Select a Protocol of **HTTPS (includes HTTP/2 and HTTP/3)**.

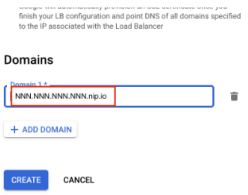
13. Under "IP address" select the named **global IP address** you created.

14. Under Certificate, select "Create a new certificate."

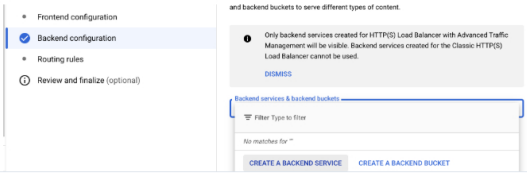


15. Give the certificate a name (ex. "summarizer-app-cert") and select "Create a Google-managed certificate."

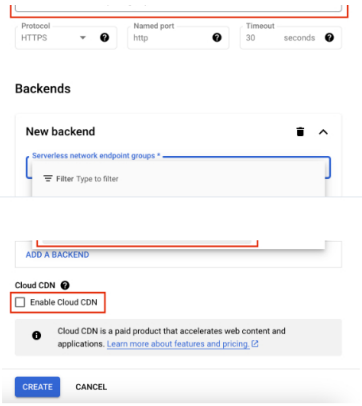
    In a real deployment setting, you would enter your domain (ex. summarizer.com) or a subdomain (ex. summarizer.google.com) and then create A (or AAAA if using IPv6) DNS records to forward traffic to the load balancer's static IP address. For this demo, we will use a handy website, nip.io, which routes traffic to an IP provided to it as a subdomain. So copy the static IP address you created earlier add ".nip.io" to the end of it so that it reads like "NNN.NNN.NNN.NNN.nip.io" with your IP address digits replacing the Ns. Paste that into the "Domain 1" box of your certificate creation model. When we complete our setup, you will be able to access your app at "https://NNN.NNN.NNN.NNN.nip.io" (again with your IP address digits replacing the Ns). Select "Create."



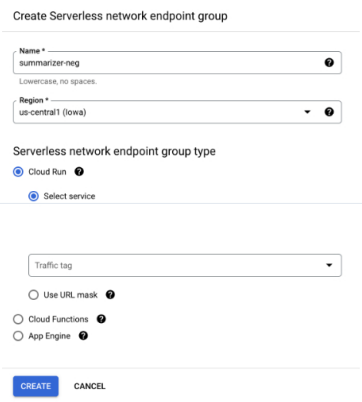
16. Back in the load balancer's configuration, select "Backend configuration" then the "Backend services & backend buckets" dropdown and then "Create a backend service."



17. Name our backend (ex. "summ-backend"). For a backend type, select "Serverless network endpoint group", then under the "New backend" header, select the "Serverless network endpoint groups" dropdown and then "**Create serverless network endpoint group**".



18. Enter a name for your Serverless NEG (ex. "summarizer-app-neg") and select `us-west1` as your Region. For type, select **Cloud Run**, then select our deployed service and select "Create".

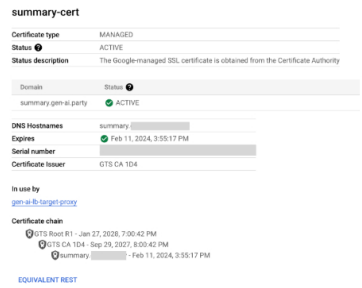


19. Back in the "Create a backend service" panel, **disable the "Enable Cloud CDN"** option (which isn't compatible with Identity-Aware Proxy).

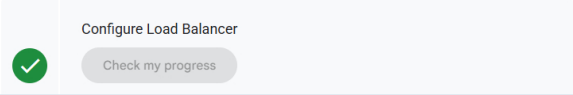
20. Select "**Create**" to finalize creating the backend.

21. Select "**Create**" one more time to create your load balancer.

22. On the Load balancing services page, we can see a spinning icon as the load balancer is created. Once it is created, we can click on the load balancer's name to see more details. In the Frontend section, we can select the certificate's name (under "Certificate") to see our SSL certificate's status. This should move from "PROVISIONING" to "ACTIVE" within about 10-20 minutes (though in some cases it could take longer). You can proceed a little further with the lab while this



Click *Check my progress* to verify the objective.

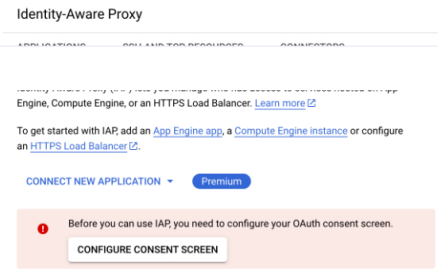


Task 6. Enable Identity-Aware Proxy

You can now configure Identity-Aware Proxy to grant access to the application.

1. Search for Identity-Aware Proxy (or "IAP") in the Console to navigate to it, then select "Enable API". Once the API is enabled, select "Go to Identity-Aware Proxy".

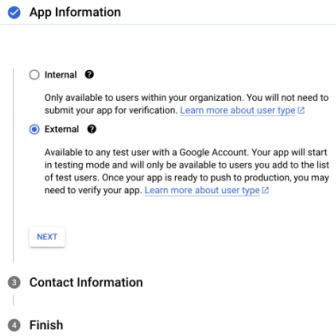
2. You will be prompted to "Configure Consent Screen". A consent screen is what is shown to a user to display which elements of their information are requested by the app and to give them a chance to agree to that or not. Select "Configure Consent Screen".



3. On the "OAuth Overview" page, click on **GET STARTED**, then under "App Information" enter your app with a name (ex. "Summarizer App")

4. Provide a user support email address (you can use your Qwiklabs-generated student ID email).

5. Select a Audience Type of "External", which you would do for a real deployment if you expect to share the app with users outside of your organization's domain. Select "Next".



6. Add your Qwiklabs student email or your own email address as the "Contact Information" email address. Click "Save and continue". You can also click "Next".

7. Accept the terms and condition for "Google API Services: User Data Policy." then click **Continue**.

8. Click on **CREATE**.

10. Under **Authorized redirect URIs** click on **+ ADD URL** an enter URL "https://NNN.NNN.NNN.NNN.nip.io" then click **Create**.

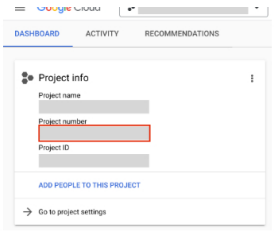
11. On the "OAuth consent screen" page, navigate to **Audience**, under Publishing Status, select "Publish app" and then click "Confirm". This will allow you to add any Google identity to which you grant the "IAP-secured Web App User" role to access the app, rather than additionally needing to add them as a test user on the OAuth consent screen.

12. Navigate back to the "Load Balancing" dashboard, select your load balancer, and then the Certificate name. If this is not yet ACTIVE, we will need to wait until it reaches ACTIVE status. Take a break and refresh occasionally.

13. When the certificate is **ACTIVE**, navigate back to Identity-Aware Proxy by searching "IAP" at the top of the Console.

14. Toggle on IAP protection of our backend service. The backend service may show a status of Error before you enable IAP, but enabling it should complete its configuration. You will be prompted to review configuration requirements, which you may want to do, and then select the checkbox confirming your understanding

15. Return to Cloud Run, select your service, and then the **Security** tab. Set the Authentication setting to **Require authentication** and select "Save." This is important to prevent your service from being accessible at its Cloud Run link directly, which would bypass the IAP protection. This change may take a few minutes to become active.

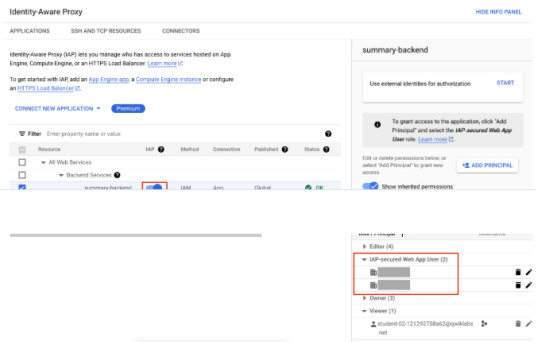


16. We need to grant the IAP service account a role within IAM. You will need your Project Number (a number that is different from your Project ID). You can find it by navigating to the Cloud Console's upper-left menu and selecting **Cloud**

replacing [PROJECT-NUMBER] with your project NUMBER: `service-[PROJECT-NUMBER]@gcp-sa-iap.iam.gserviceaccount.com`

17. Navigate to IAM, select **+ Grant Access**, and grant that service account the IAM Role of `Cloud Run Invoker`.

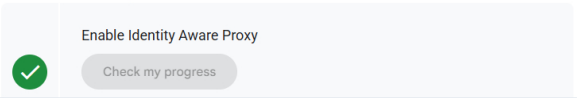
18. You can now grant access to individual users (betty@example.com), Google Groups (marketing@example.com), or entire domains (example.com) by granting them the "IAP-secured Web App User" IAM role. This role can be granted on the project overall, or on a single selected Backend Service within IAP. To use this fine-grained control, navigate back to the IAP dashboard, and select our backend service. Then select **+Add Principal** in the panel on the right.



19. Add a Google Identity of yours as a user with the "IAP-secured Web App User" role.

20. You may see an "Error: Forbidden" message for about the first 5 minutes, but after that users with the "IAP-secured Web App User" role on the Project or IAP backend service should be able to access the app via the domain we listed on our Load Balancer certificate (https://NNN.NNN.NNN.NNN.nip.io with your load balancer's IP address replacing the Ns). Make sure to use `https` and not simply `http` at the start of the URL.

Click *Check my progress* to verify the objective.



Congratulations

You have now completed the lab! In this lab, you deployed a Cloud Run app that can make calls to a Vertex AI Language Model. You also protected your app from unauthorized users using Identity-Aware Proxy.

Next steps

- Check out the Generative AI on Vertex AI documentation.
- Learn more about Generative AI on the Google Cloud Tech YouTube channel.

Google Cloud Training & Certification

...helps you make the most of Google Cloud technologies. Our classes include technical skills and best practices to help you get up to speed quickly and continue your learning journey. We offer fundamental to advanced level training, with on-demand, live, and

virtual options to suit your busy schedule. Certifications help you validate and prove your skill and expertise in Google Cloud technologies.

Manual Last Updated March 21, 2025

Lab Last Tested March 21, 2025