every two weeks
#gcpbytes

contact@gcpbytes.com.au

GCP BYTES

Listen on Apple Podcasts | Listen on Spotify | LISTEN ON YouTube

Stephen Bancroft

Dave Wall

Who am I?

https://www.ianbrown.id.au

https://g.dev/ianbrown

https://www.linkedin.com/in/ian-brown-au/

https://github.com/ianbrown78

A Gemini Driven Slackbot

Why?

Why?

Well, why not?

Why?

Well, why not?

- Gemini is a hot topic
- Slack is the industry default messaging platform
- Gemini + Slack + Bot = Awesomeness

Caveats

Caveats

1. I am NOT a python developer!

Caveats

1. I am NOT a Python developer!
2. Also, I am NOT a Python developer!

Caveats

1. I am NOT a Python developer!
2. Also, I am NOT a Python developer!
3. Lastly, I am NOT a Python developer!

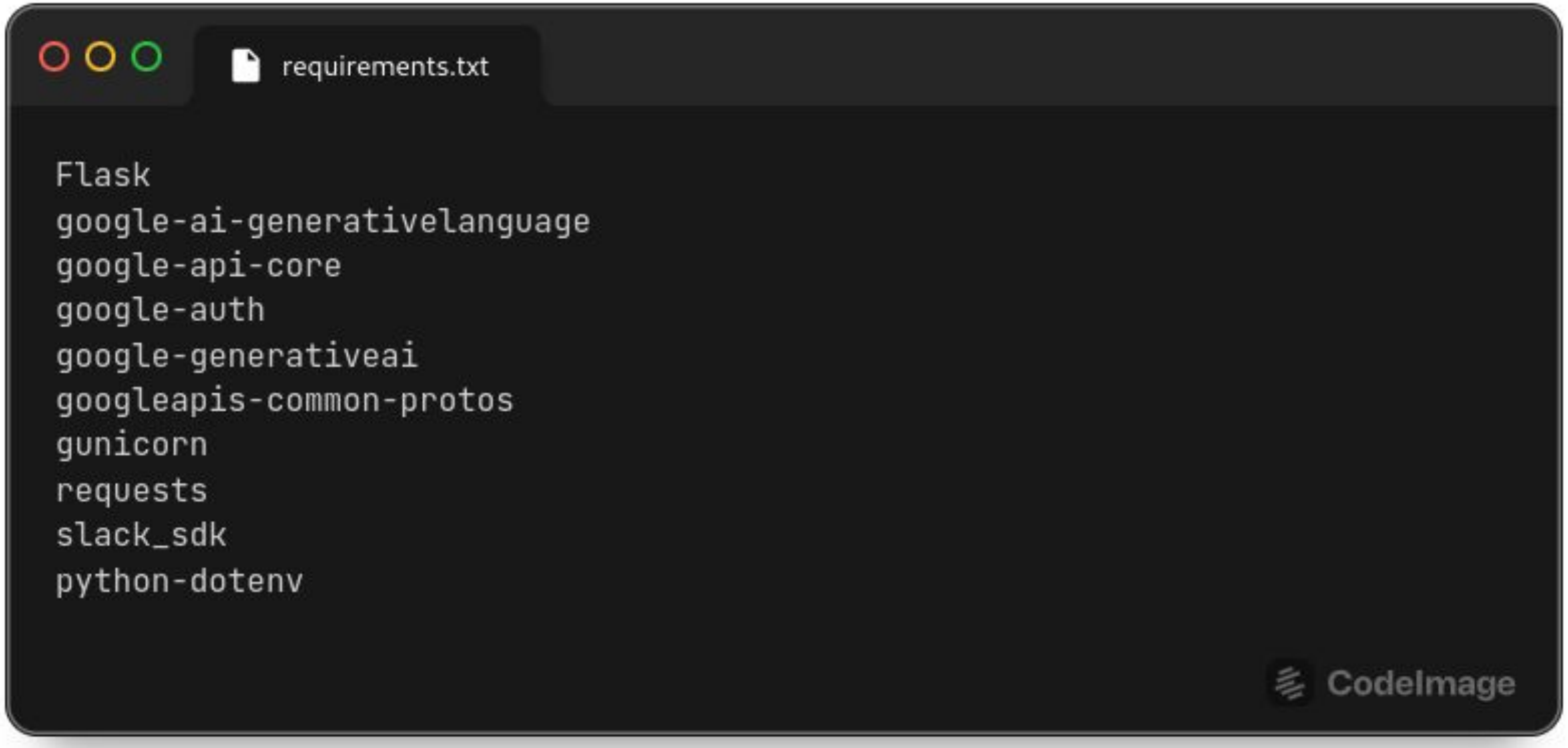
Code Time!

```
//fires the appear event when appropriate
var check = function() {
    //is the element hidden?
    if (!t.is(':visible')) {
        //it became hidden
        t.appeared = false;
        return;
    }

    //is the element inside the visible window?
    var a = w.scrollLeft();
    var b = w.scrollTop();
    var o = t.offset();
    var x = o.left;
    var y = o.top;

    var ax = settings.accX;
    var ay = settings.accY;
    var th = t.height();
    var wh = w.height();
    var tw = t.width();
    var ww = w.width();

    if (y + th + ay >= b &&
        y <= b + wh + ay &&
        x + tw + ax >= a &&
        x <= a + ww + ax) {

        //trigger the custom event
        if (!t.appeared) t.trigger('appear', settings.data);

    } else {

        //it scrolled out of view
        t.appeared = false;
    }
};

//create a modified fn with some additional logic
var modifiedFn = function() {

    //mark the element as visible
    t.appeared = true;

    //is this supposed to happen only once?
    if (settings.one) {

        //remove the check
        w.unbind('scroll', check);
        var i = $.inArray(check, $.fn.appear.checks);
        if (i >= 0) $.fn.appear.checks.splice(i, 1);
    }

    //trigger the original fn
    fn.apply(this, arguments);
};

//bind the modified fn to the element
    one) t.one('appear', settings.data, modifiedFn);
         ettings.data, modifiedFn);
```

Code Time!

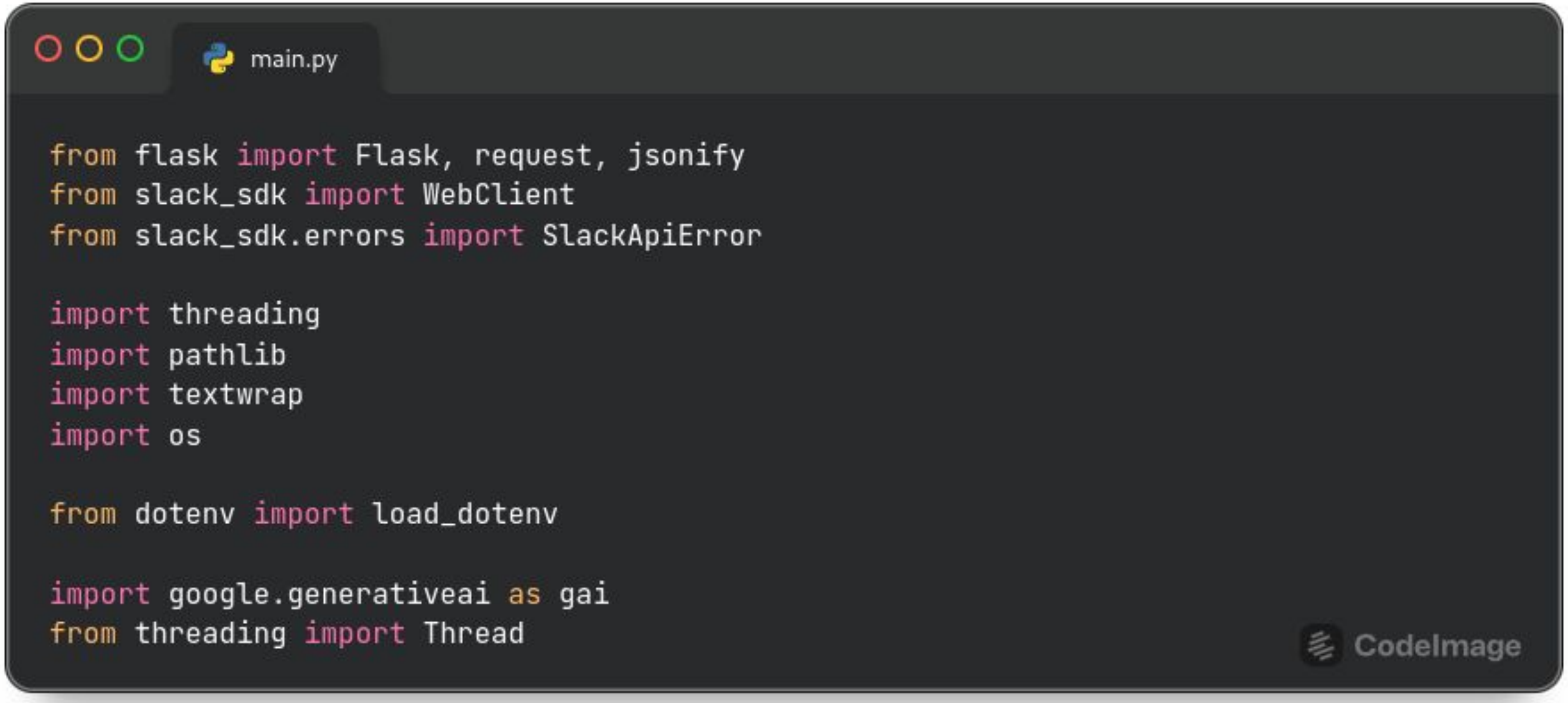
1. Install python3-venv for your OS
2. Create a directory for our code
3. Create the virtual environment
4. Activate the venv

Code Time!


requirements.txt

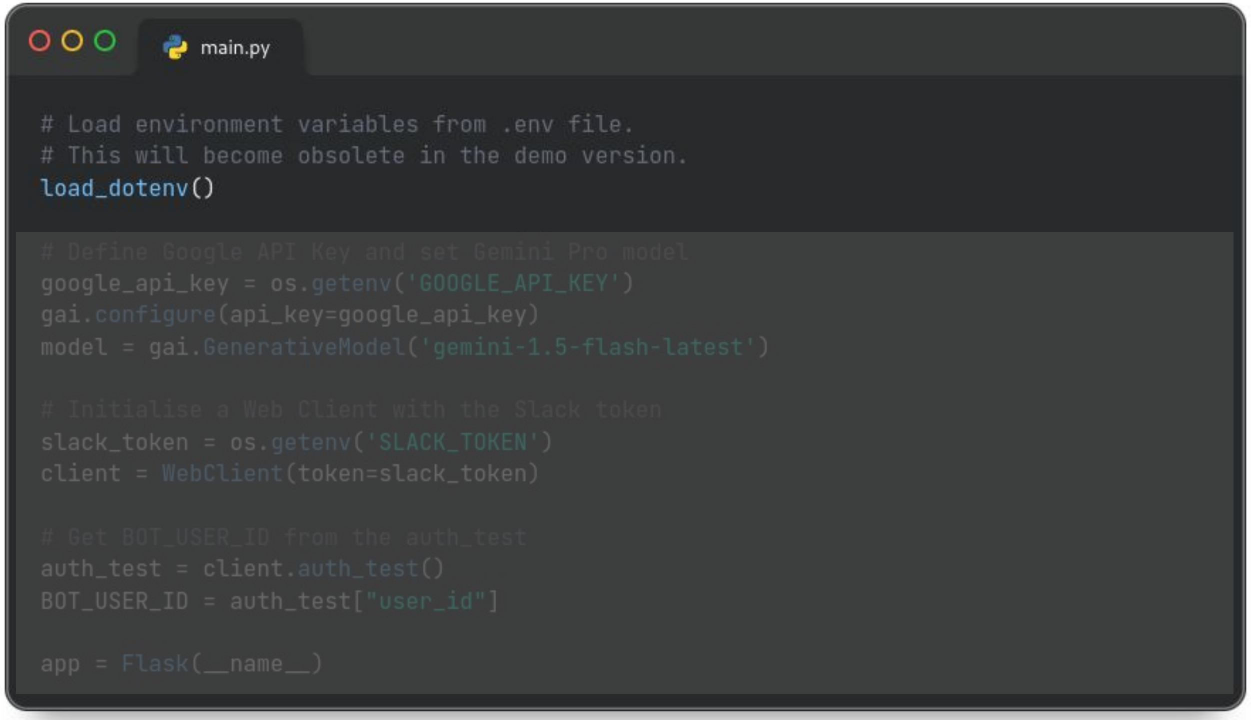
```
Flask
google-ai-generativelanguage
google-api-core
google-auth
google-generativeai
googleapis-common-protos
gunicorn
requests
slack_sdk
python-dotenv
```

Code Time!


main.py

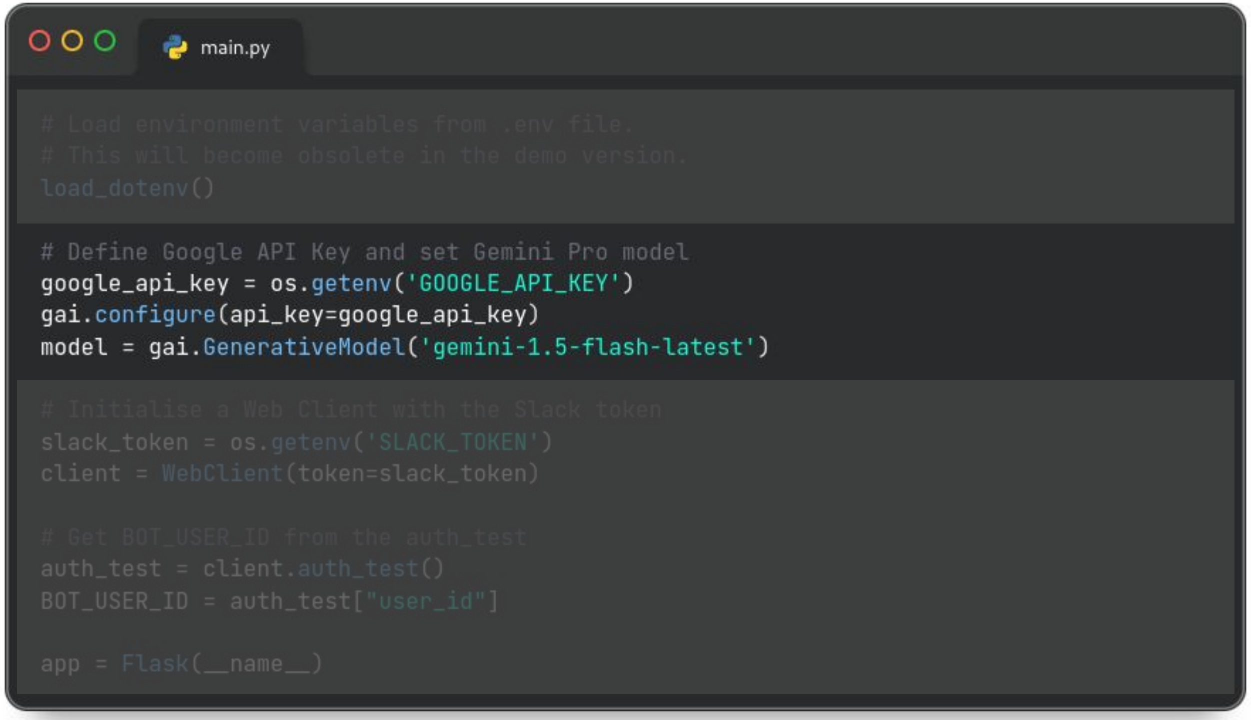
```python
from flask import Flask, request, jsonify
from slack_sdk import WebClient
from slack_sdk.errors import SlackApiError

import threading
import pathlib
import textwrap
import os

from dotenv import load_dotenv

import google.generativeai as gai
from threading import Thread
```

Code Time!


main.py

```python
# Load environment variables from .env file.
# This will become obsolete in the demo version.
load_dotenv()

# Define Google API Key and set Gemini Pro model
google_api_key = os.getenv('GOOGLE_API_KEY')
gai.configure(api_key=google_api_key)
model = gai.GenerativeModel('gemini-1.5-flash-latest')

# Initialise a Web Client with the Slack token
slack_token = os.getenv('SLACK_TOKEN')
client = WebClient(token=slack_token)

# Get BOT_USER_ID from the auth_test
auth_test = client.auth_test()
BOT_USER_ID = auth_test["user_id"]

app = Flask(__name__)
```

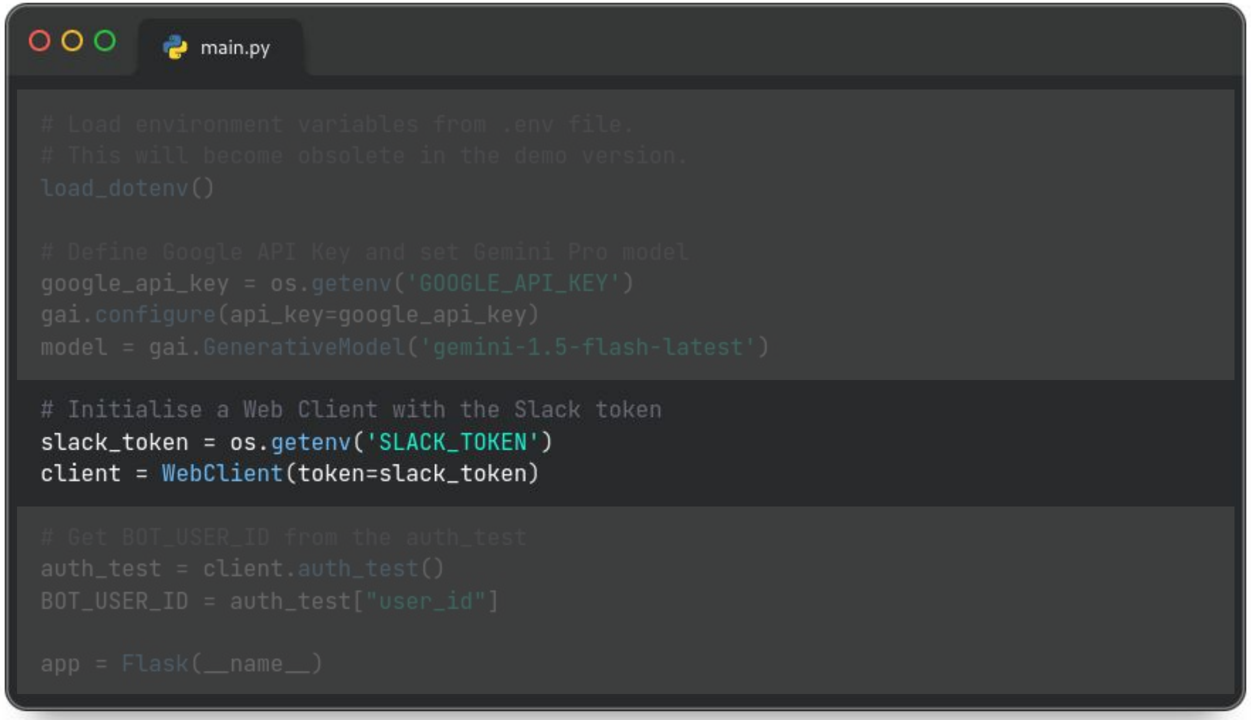
Code Time!



main.py

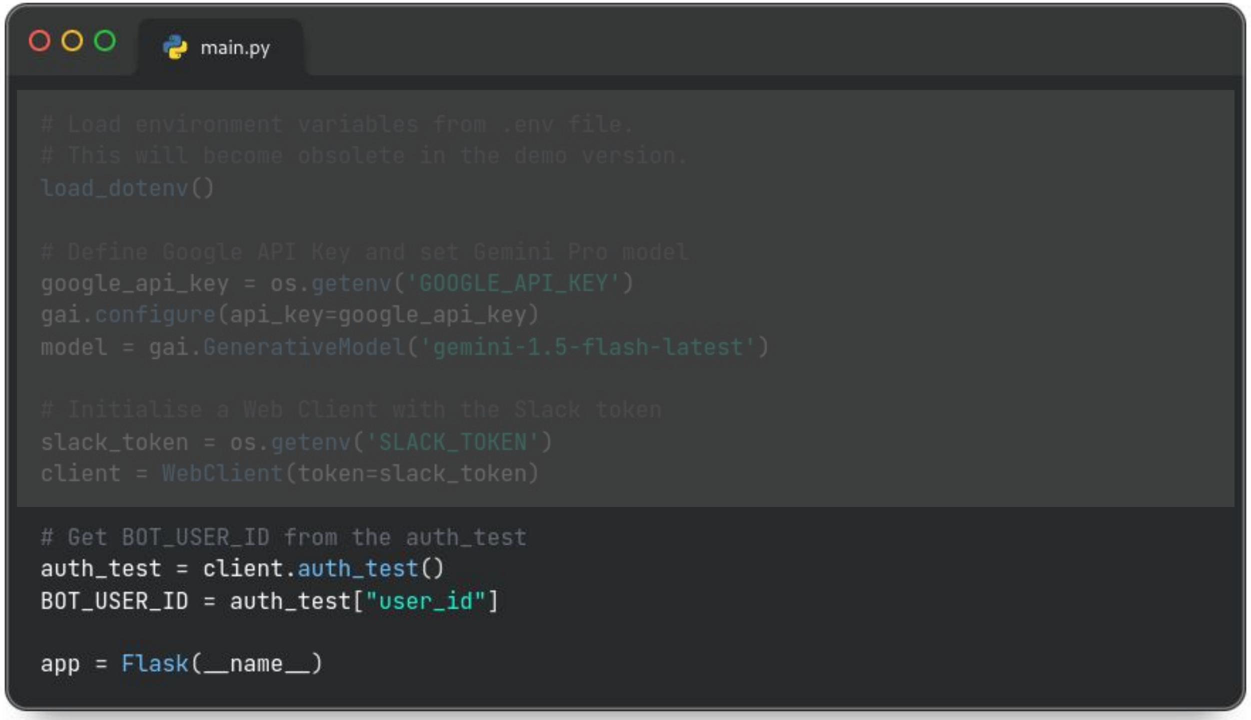
```python
# Load environment variables from .env file.
# This will become obsolete in the demo version.
load_dotenv()

# Define Google API Key and set Gemini Pro model
google_api_key = os.getenv('GOOGLE_API_KEY')
gai.configure(api_key=google_api_key)
model = gai.GenerativeModel('gemini-1.5-flash-latest')

# Initialise a Web Client with the Slack token
slack_token = os.getenv('SLACK_TOKEN')
client = WebClient(token=slack_token)

# Get BOT_USER_ID from the auth_test
auth_test = client.auth_test()
BOT_USER_ID = auth_test["user_id"]

app = Flask(__name__)
```

Code Time!


main.py

```python
# Load environment variables from .env file.
# This will become obsolete in the demo version.
load_dotenv()

# Define Google API Key and set Gemini Pro model
google_api_key = os.getenv('GOOGLE_API_KEY')
gai.configure(api_key=google_api_key)
model = gai.GenerativeModel('gemini-1.5-flash-latest')

# Initialise a Web Client with the Slack token
slack_token = os.getenv('SLACK_TOKEN')
client = WebClient(token=slack_token)

# Get BOT_USER_ID from the auth_test
auth_test = client.auth_test()
BOT_USER_ID = auth_test["user_id"]

app = Flask(__name__)
```

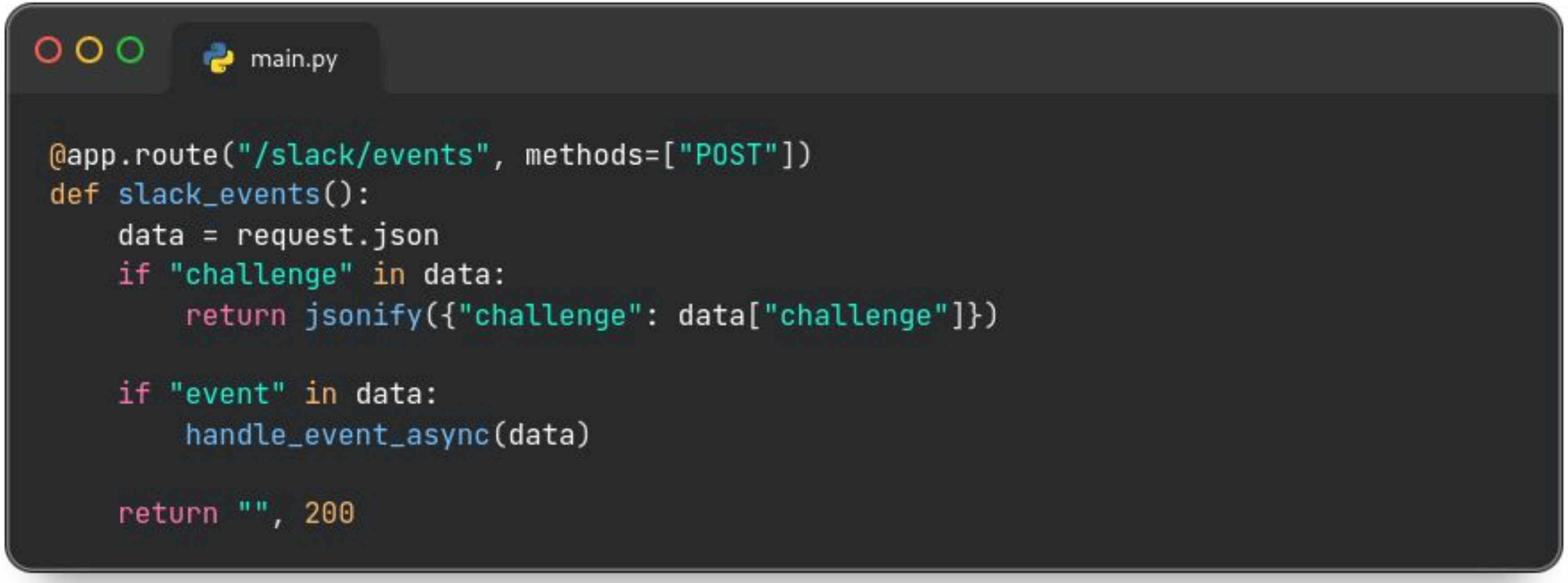
Code Time!



main.py

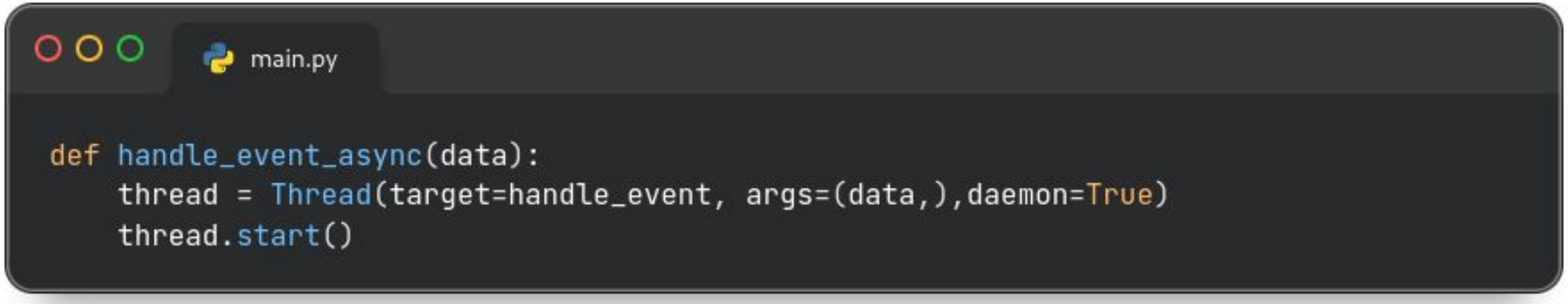
```python
# Load environment variables from .env file.
# This will become obsolete in the demo version.
load_dotenv()

# Define Google API Key and set Gemini Pro model
google_api_key = os.getenv('GOOGLE_API_KEY')
gai.configure(api_key=google_api_key)
model = gai.GenerativeModel('gemini-1.5-flash-latest')

# Initialise a Web Client with the Slack token
slack_token = os.getenv('SLACK_TOKEN')
client = WebClient(token=slack_token)

# Get BOT_USER_ID from the auth_test
auth_test = client.auth_test()
BOT_USER_ID = auth_test["user_id"]

app = Flask(__name__)
```

Code Time!


main.py

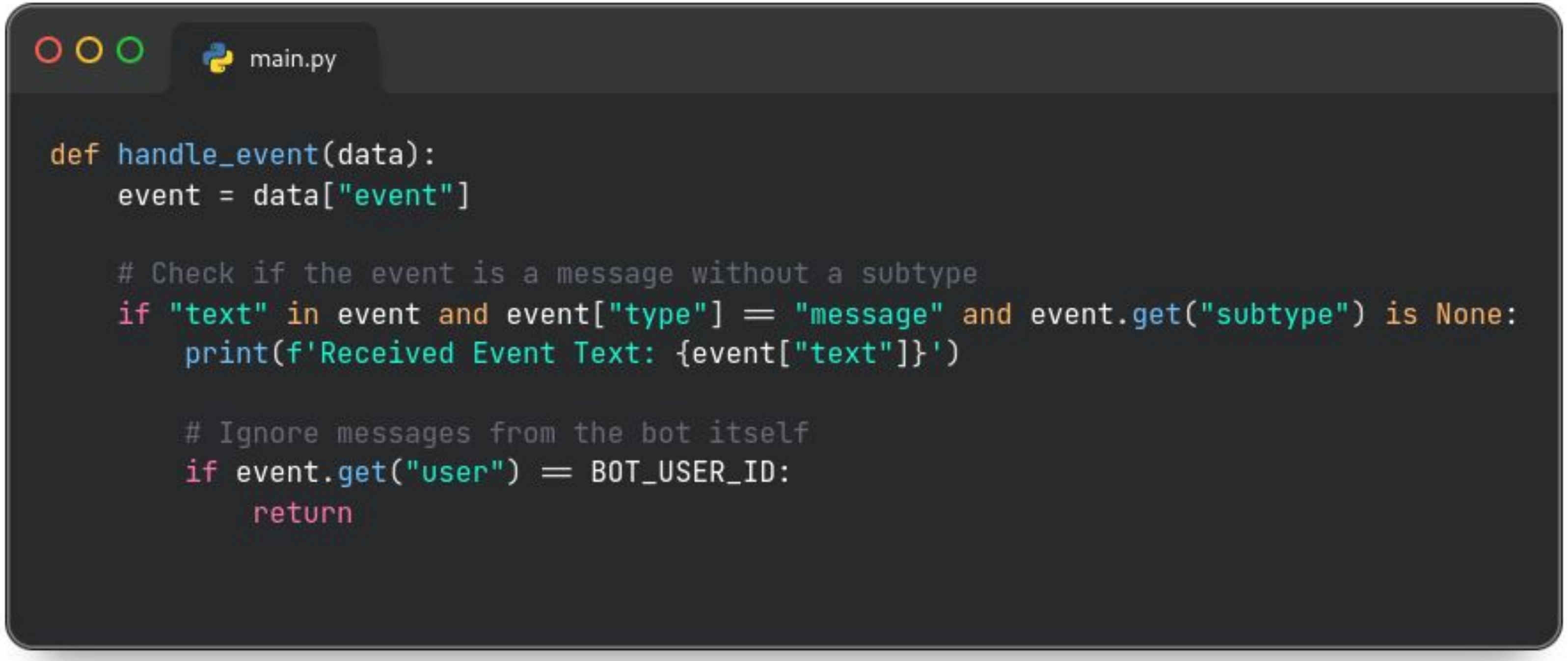
```
@app.route("/slack/events", methods=["POST"])
def slack_events():
    data = request.json
    if "challenge" in data:
        return jsonify({"challenge": data["challenge"]})

    if "event" in data:
        handle_event_async(data)

    return "", 200
```

Code Time!



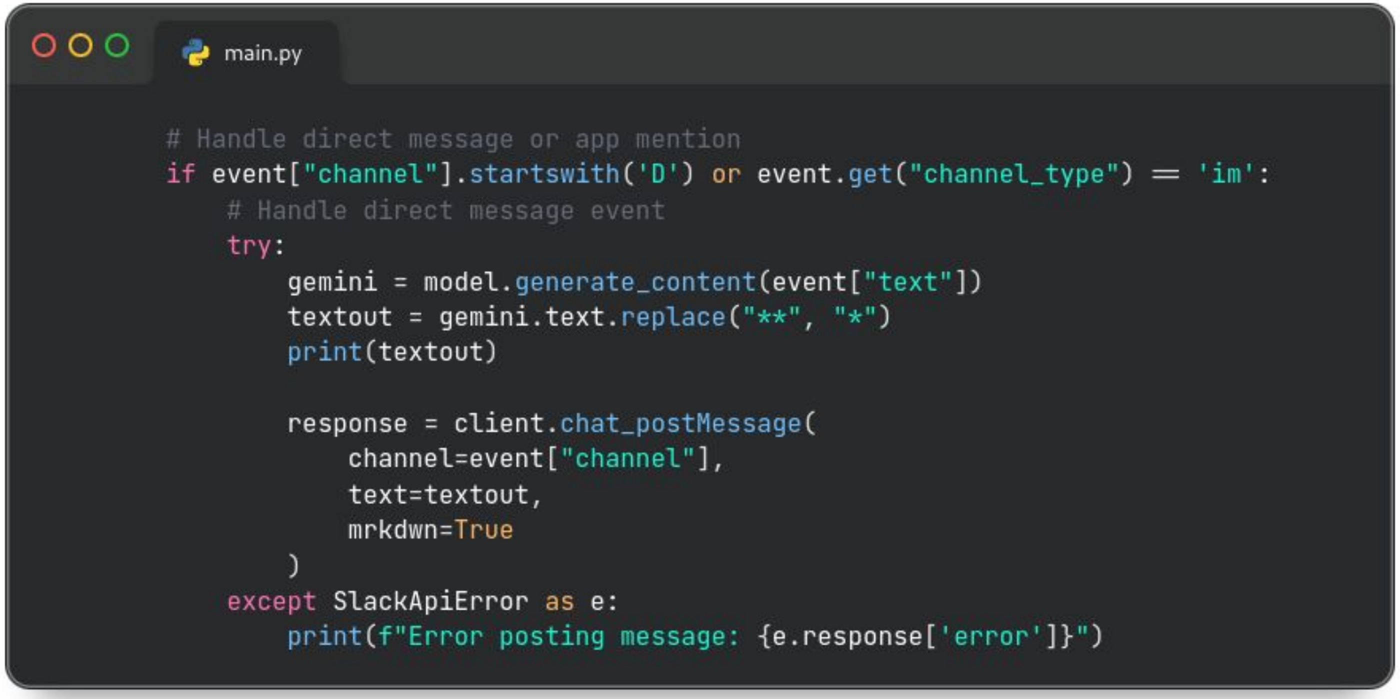
```python
def handle_event_async(data):
    thread = Thread(target=handle_event, args=(data,),daemon=True)
    thread.start()
```

Code Time!



```python
def handle_event(data):
    event = data["event"]

    # Check if the event is a message without a subtype
    if "text" in event and event["type"] == "message" and event.get("subtype") is None:
        print(f'Received Event Text: {event["text"]}')

        # Ignore messages from the bot itself
        if event.get("user") == BOT_USER_ID:
            return
```

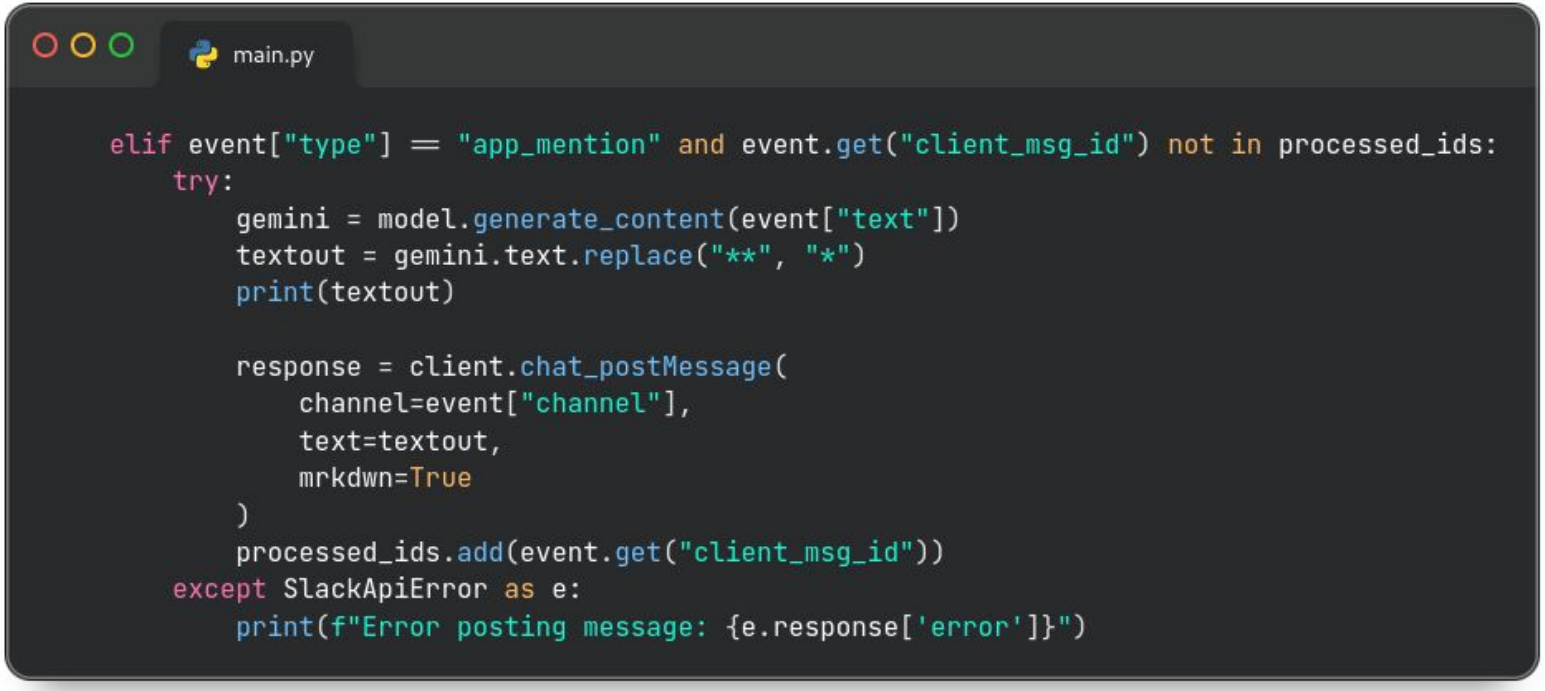
Code Time!



main.py

```python
# Handle direct message or app mention
if event["channel"].startswith('D') or event.get("channel_type") == 'im':
    # Handle direct message event
    try:
        gemini = model.generate_content(event["text"])
        textout = gemini.text.replace("**", "*")
        print(textout)

        response = client.chat_postMessage(
            channel=event["channel"],
            text=textout,
            mrkdwn=True
        )
    except SlackApiError as e:
        print(f"Error posting message: {e.response['error']}")
```

Code Time!


main.py

```python
elif event["type"] == "app_mention" and event.get("client_msg_id") not in processed_ids:
    try:
        gemini = model.generate_content(event["text"])
        textout = gemini.text.replace("**", "*")
        print(textout)

        response = client.chat_postMessage(
            channel=event["channel"],
            text=textout,
            mrkdwn=True
        )
        processed_ids.add(event.get("client_msg_id"))
    except SlackApiError as e:
        print(f"Error posting message: {e.response['error']}")
```

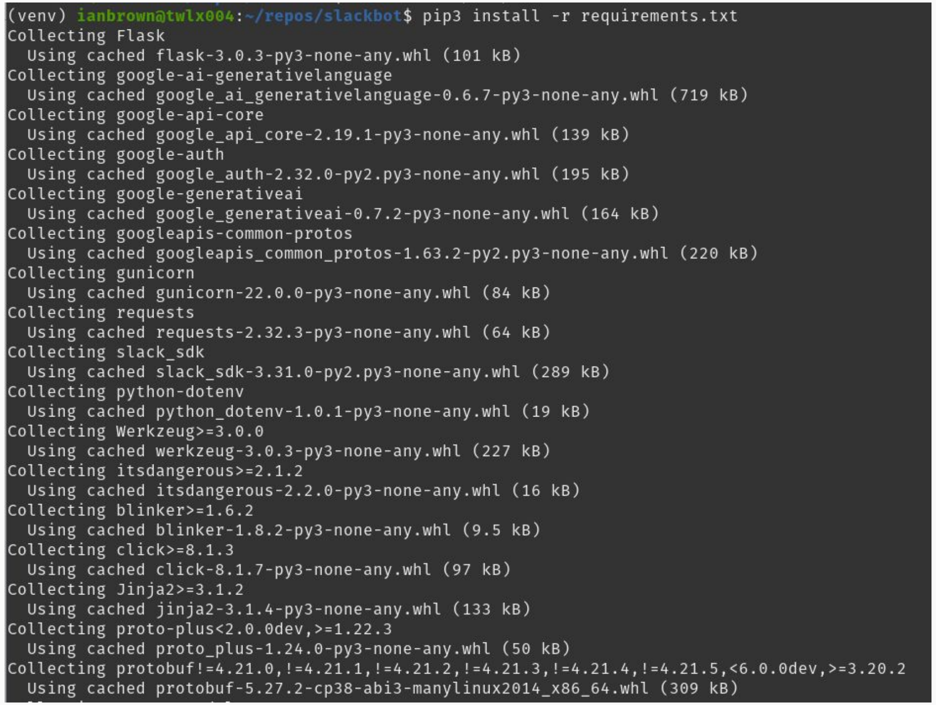
Code Time!

Install the requirements

```
pip3 install -r requirements.txt
```

Build the container

```
docker build -t slackbot -f Dockerfile .
docker push slackbot
```



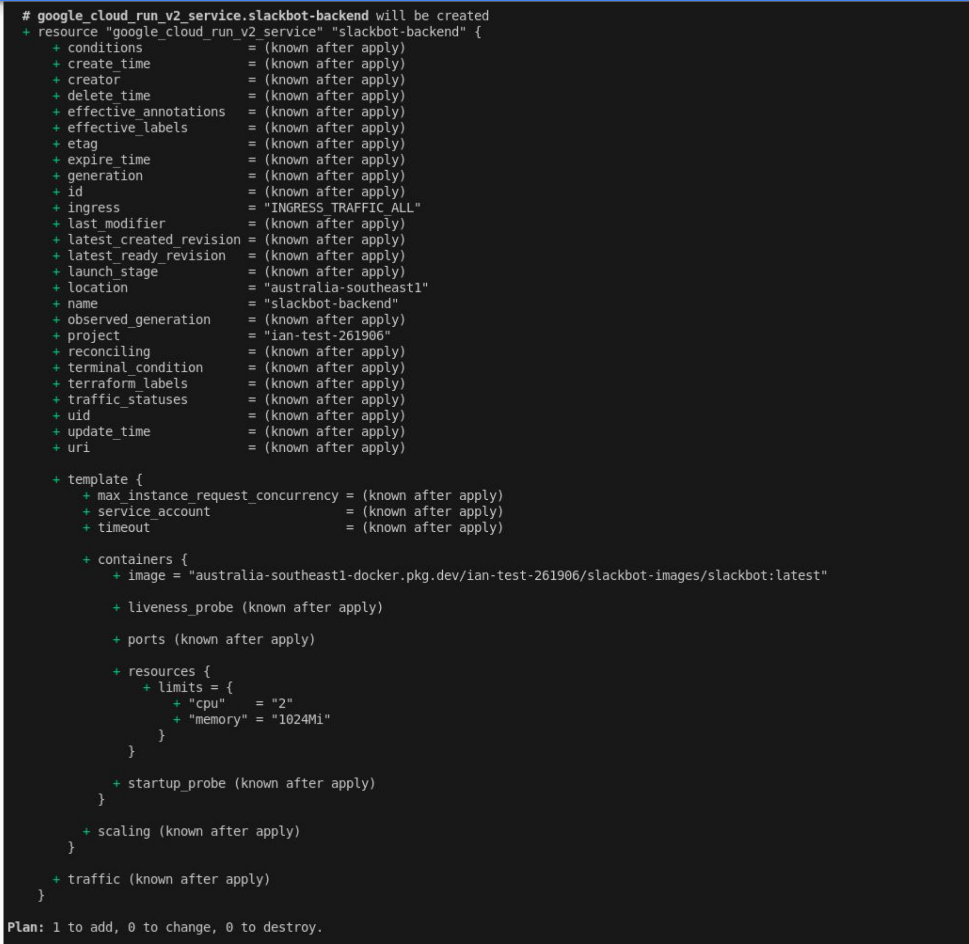
```
(venv) ianbrown@twlx004:~/repos/slackbot$ pip3 install -r requirements.txt
Collecting Flask
  Using cached flask-3.0.3-py3-none-any.whl (101 kB)
Collecting google-ai-generativelanguage
  Using cached google_ai_generativelanguage-0.6.7-py3-none-any.whl (719 kB)
Collecting google-api-core
  Using cached google_api_core-2.19.1-py3-none-any.whl (139 kB)
Collecting google-auth
  Using cached google_auth-2.32.0-py2.py3-none-any.whl (195 kB)
Collecting google-generativeai
  Using cached google_generativeai-0.7.2-py3-none-any.whl (164 kB)
Collecting googleapis-common-protos
  Using cached googleapis_common_protos-1.63.2-py2.py3-none-any.whl (220 kB)
Collecting gunicorn
  Using cached gunicorn-22.0.0-py3-none-any.whl (84 kB)
Collecting requests
  Using cached requests-2.32.3-py3-none-any.whl (64 kB)
Collecting slack_sdk
  Using cached slack_sdk-3.31.0-py2.py3-none-any.whl (289 kB)
Collecting python-dotenv
  Using cached python_dotenv-1.0.1-py3-none-any.whl (19 kB)
Collecting Werkzeug>=3.0.0
  Using cached werkzeug-3.0.3-py3-none-any.whl (227 kB)
Collecting itsdangerous>=2.1.2
  Using cached itsdangerous-2.2.0-py3-none-any.whl (16 kB)
Collecting blinker>=1.6.2
  Using cached blinker-1.8.2-py3-none-any.whl (9.5 kB)
Collecting click>=8.1.3
  Using cached click-8.1.7-py3-none-any.whl (97 kB)
Collecting Jinja2>=3.1.2
  Using cached jinja2-3.1.4-py3-none-any.whl (133 kB)
Collecting proto-plus<2.0.0dev,>=1.22.3
  Using cached proto_plus-1.24.0-py3-none-any.whl (50 kB)
Collecting protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.2
  Using cached protobuf-5.27.2-cp38-abi3-manylinux2014_x86_64.whl (309 kB)
```

Code Time!

Create your GCP project

TF apply

Build and push container image



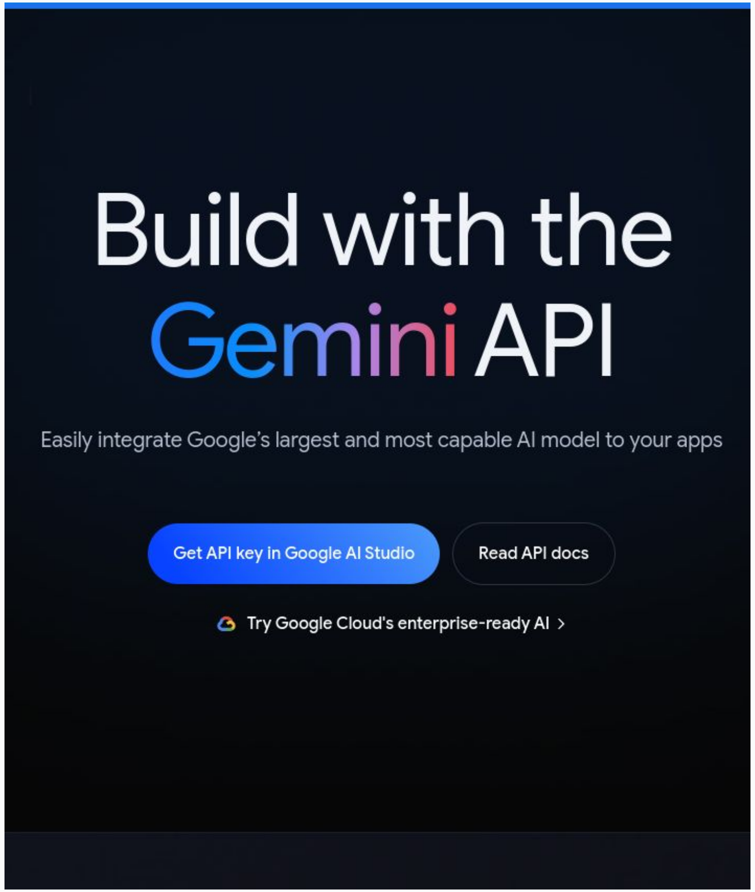
```
  # google_cloud_run_v2_service.slackbot-backend will be created
  + resource "google_cloud_run_v2_service" "slackbot-backend" {
      + conditions              = (known after apply)
      + create_time             = (known after apply)
      + creator                 = (known after apply)
      + delete_time             = (known after apply)
      + effective_annotations   = (known after apply)
      + effective_labels        = (known after apply)
      + etag                    = (known after apply)
      + expire_time             = (known after apply)
      + generation              = (known after apply)
      + id                      = (known after apply)
      + ingress                 = "INGRESS_TRAFFIC_ALL"
      + last_modifier           = (known after apply)
      + latest_created_revision = (known after apply)
      + latest_ready_revision   = (known after apply)
      + launch_stage            = (known after apply)
      + location                = "australia-southeast1"
      + name                    = "slackbot-backend"
      + observed_generation     = (known after apply)
      + project                 = "ian-test-261906"
      + reconciling             = (known after apply)
      + terminal_condition      = (known after apply)
      + terraform_labels        = (known after apply)
      + traffic_statuses        = (known after apply)
      + uid                     = (known after apply)
      + update_time             = (known after apply)
      + uri                     = (known after apply)

      + template {
          + max_instance_request_concurrency = (known after apply)
          + service_account                  = (known after apply)
          + timeout                          = (known after apply)

          + containers {
              + image = "australia-southeast1-docker.pkg.dev/ian-test-261906/slackbot-images/slackbot:latest"

              + liveness_probe (known after apply)

              + ports (known after apply)

              + resources {
                  + limits = {
                      + "cpu"    = "2"
                      + "memory" = "1024Mi"
                    }
                }

              + startup_probe (known after apply)
            }

          + scaling (known after apply)
        }

      + traffic (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Code Time!

Setup Gemini AI

Add the token to the GOOGLE_API_KEY secret in secrets manager.

https://ai.google.dev/



Build with the Gemini API

Easily integrate Google's largest and most capable AI model to your apps

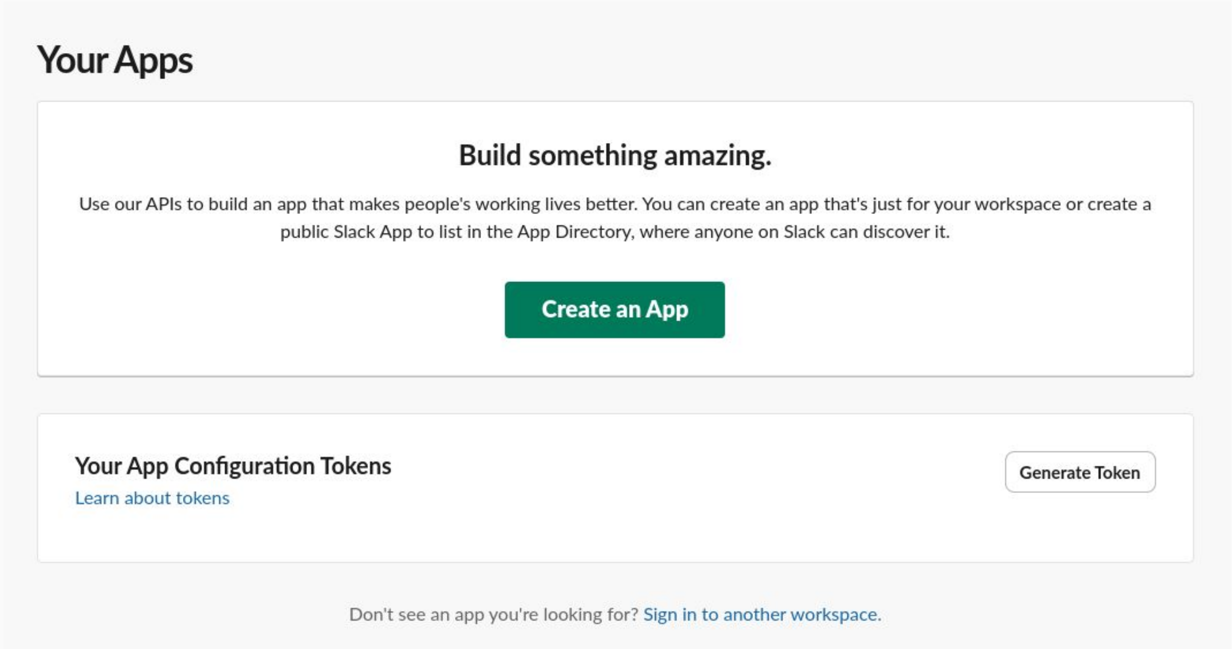
Get API key in Google AI Studio
Read API docs
Try Google Cloud's enterprise-ready AI ›

Code Time!

1. Create the Slack App

Add the token to the SLACK_TOKEN secrets manager

https://api.slack.com/apps



Your Apps

Build something amazing.

Use our APIs to build an app that makes people's working lives better. You can create an app that's just for your workspace or create a public Slack App to list in the App Directory, where anyone on Slack can discover it.

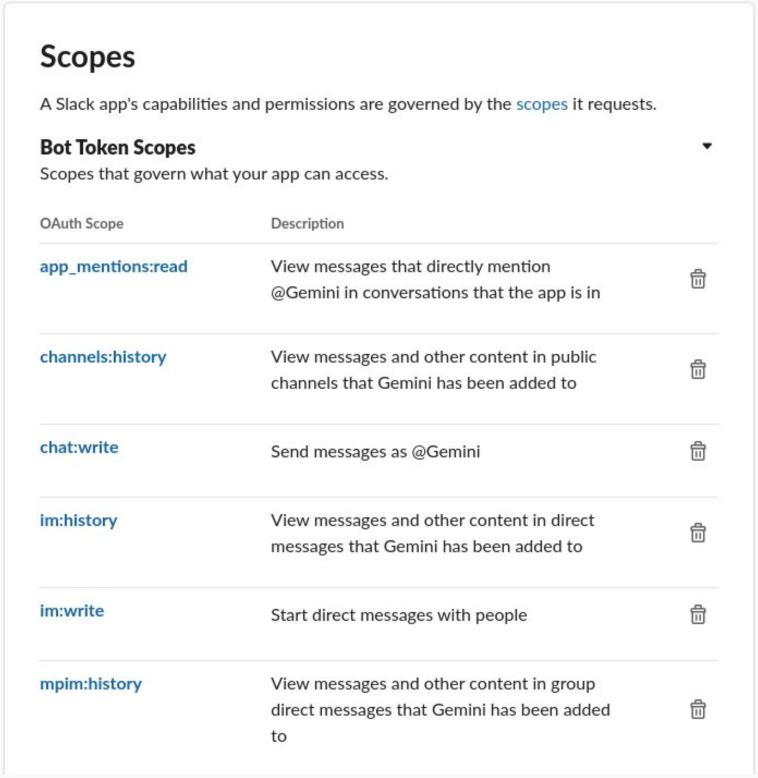
Your App Configuration Tokens

Don't see an app you're looking for? Sign in to another workspace.

Code Time!

1. Create the Slack App
2. Add Scopes

https://api.slack.com/apps



Scopes

A Slack app's capabilities and permissions are governed by the scopes it requests.

Bot Token Scopes

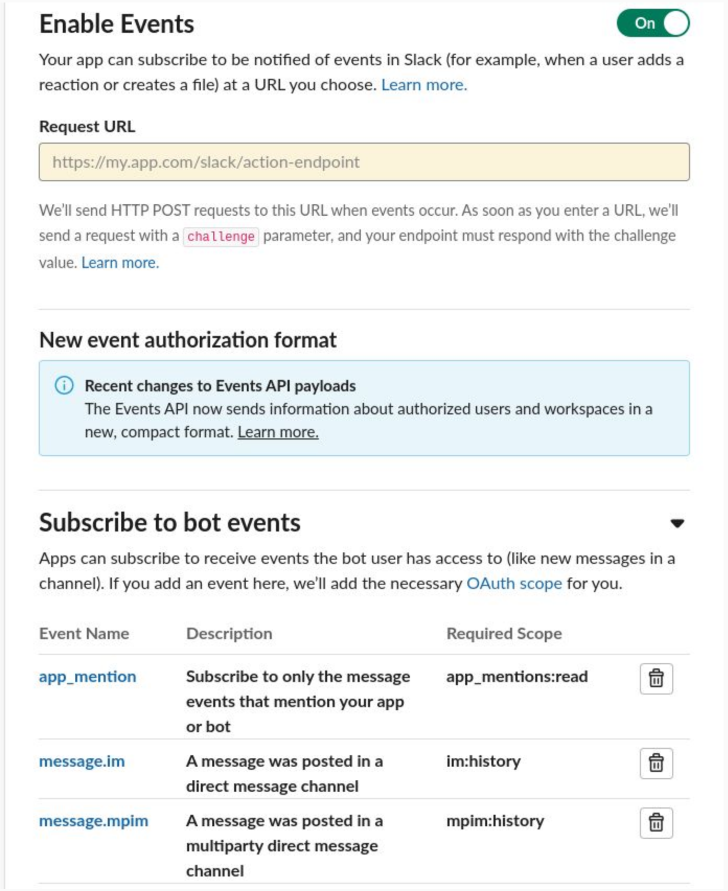
Scopes that govern what your app can access.

OAuth Scope	Description
app_mentions:read	View messages that directly mention @Gemini in conversations that the app is in
channels:history	View messages and other content in public channels that Gemini has been added to
chat:write	Send messages as @Gemini
im:history	View messages and other content in direct messages that Gemini has been added to
im:write	Start direct messages with people
mpim:history	View messages and other content in group direct messages that Gemini has been added to

Code Time!

1. Create the Slack App
2. Add Scopes
3. Add Event Subscriptions

https://api.slack.com/apps



Enable Events

Your app can subscribe to be notified of events in Slack (for example, when a user adds a reaction or creates a file) at a URL you choose. Learn more.

Request URL

https://my.app.com/slack/action-endpoint

We'll send HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a `challenge` parameter, and your endpoint must respond with the challenge value. Learn more.

New event authorization format

Recent changes to Events API payloads
The Events API now sends information about authorized users and workspaces in a new, compact format. Learn more.

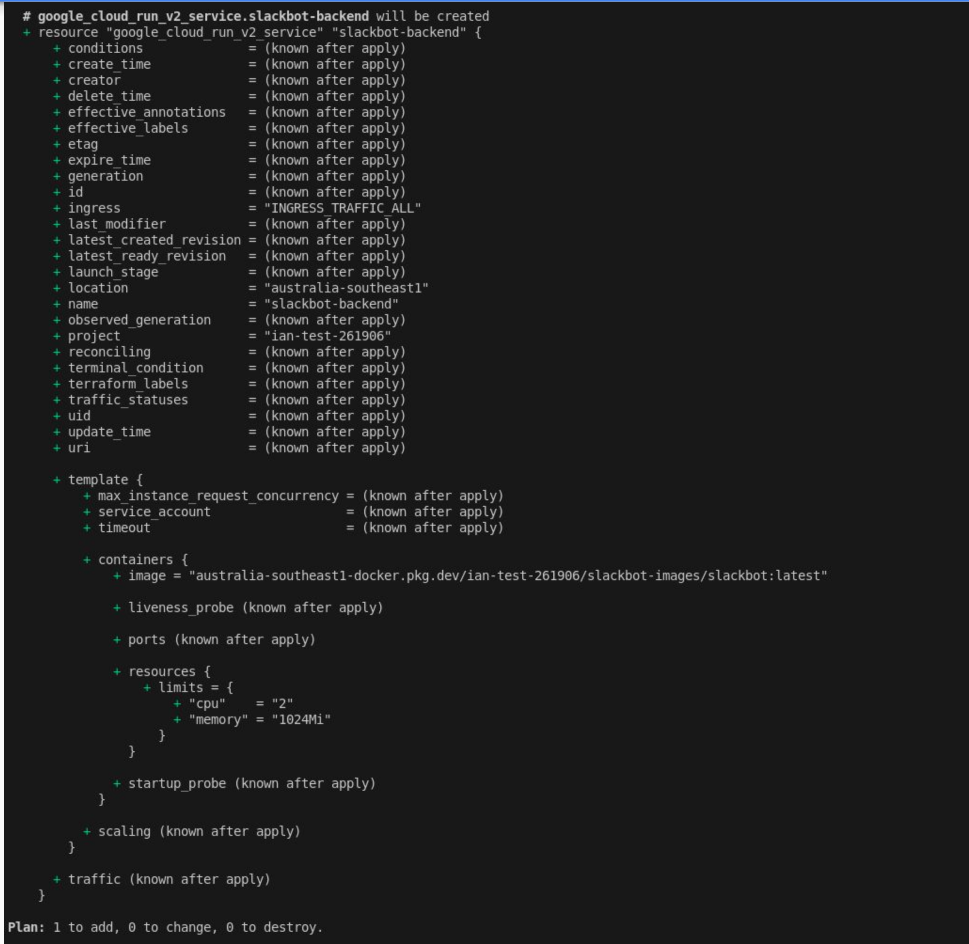
Subscribe to bot events

Apps can subscribe to receive events the bot user has access to (like new messages in a channel). If you add an event here, we'll add the necessary OAuth scope for you.

Event Name	Description	Required Scope
app_mention	Subscribe to only the message events that mention your app or bot	app_mentions:read
message.im	A message was posted in a direct message channel	im:history
message.mpim	A message was posted in a multiparty direct message channel	mpim:history

Code Time!

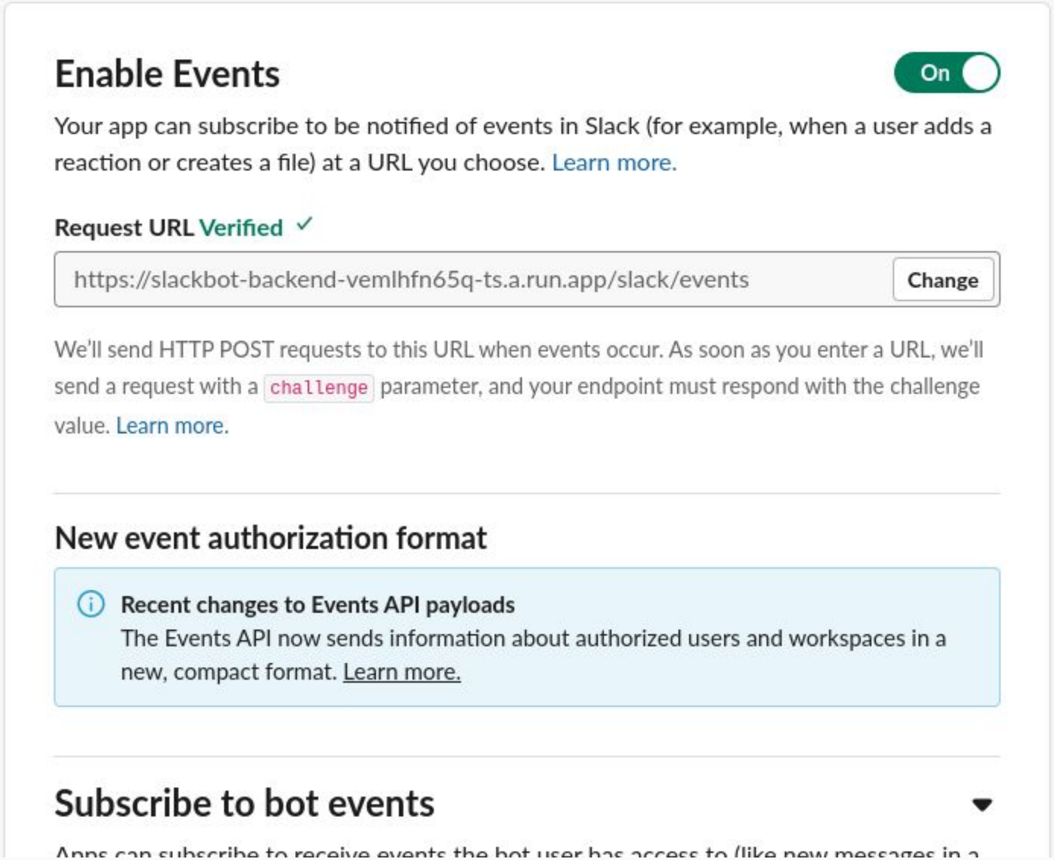
TF apply again



```
  # google_cloud_run_v2_service.slackbot-backend will be created
  + resource "google_cloud_run_v2_service" "slackbot-backend" {
      + conditions              = (known after apply)
      + create_time             = (known after apply)
      + creator                 = (known after apply)
      + delete_time             = (known after apply)
      + effective_annotations   = (known after apply)
      + effective_labels        = (known after apply)
      + etag                    = (known after apply)
      + expire_time             = (known after apply)
      + generation              = (known after apply)
      + id                      = (known after apply)
      + ingress                 = "INGRESS_TRAFFIC_ALL"
      + last_modifier           = (known after apply)
      + latest_created_revision = (known after apply)
      + latest_ready_revision   = (known after apply)
      + launch_stage            = (known after apply)
      + location                = "australia-southeast1"
      + name                    = "slackbot-backend"
      + observed_generation     = (known after apply)
      + project                 = "ian-test-261906"
      + reconciling             = (known after apply)
      + terminal_condition      = (known after apply)
      + terraform_labels        = (known after apply)
      + traffic_statuses        = (known after apply)
      + uid                     = (known after apply)
      + update_time             = (known after apply)
      + uri                     = (known after apply)

      + template {
          + max_instance_request_concurrency = (known after apply)
          + service_account                  = (known after apply)
          + timeout                          = (known after apply)

          + containers {
              + image = "australia-southeast1-docker.pkg.dev/ian-test-261906/slackbot-images/slackbot:latest"

              + liveness_probe (known after apply)

              + ports (known after apply)

              + resources {
                  + limits = {
                      + "cpu"    = "2"
                      + "memory" = "1024Mi"
                    }
                }

              + startup_probe (known after apply)
            }

          + scaling (known after apply)
        }

      + traffic (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Code Time!

Add the Request URL



Enable Events

Your app can subscribe to be notified of events in Slack (for example, when a user adds a reaction or creates a file) at a URL you choose. Learn more.

Request URL Verified ✓

https://slackbot-backend-vemlhfn65q-ts.a.run.app/slack/events

We'll send HTTP POST requests to this URL when events occur. As soon as you enter a URL, we'll send a request with a `challenge` parameter, and your endpoint must respond with the challenge value. Learn more.

New event authorization format

Recent changes to Events API payloads
The Events API now sends information about authorized users and workspaces in a new, compact format. Learn more.

Subscribe to bot events

Test Drive Time!

Test Drive Time!

Some interesting queries:

- Write me a simple python script to get the user_id from the slack api
- The trolley problem
- Another variant of the trolley problem

Test Drive Time!

Improvements:

- Ensure Cloud Run is using authenticated mode
- Use the events API in socket mode
- Handle other types of messages (files, images, etc)
- Automated pipelines for deployment
- Unit tests? Does anyone really use these…

Build your own!

https://github.com/ianbrown78/gemini-slackbot

Thanks!

LinkedIn:

https://www.linkedin.com/in/ian-brown-au/

Website:

https://www.ianbrown.id.au/

G.Dev:

https://g.dev/ianbrown