Connecting an AI Agent to WhatsApp

How to process WhatsApp messages using LangGraph and Cloud Run

MIGUEL OTERO PEDRIDO
MAR 12, 2025

19 1 2 Sl

The day is here, my friend.

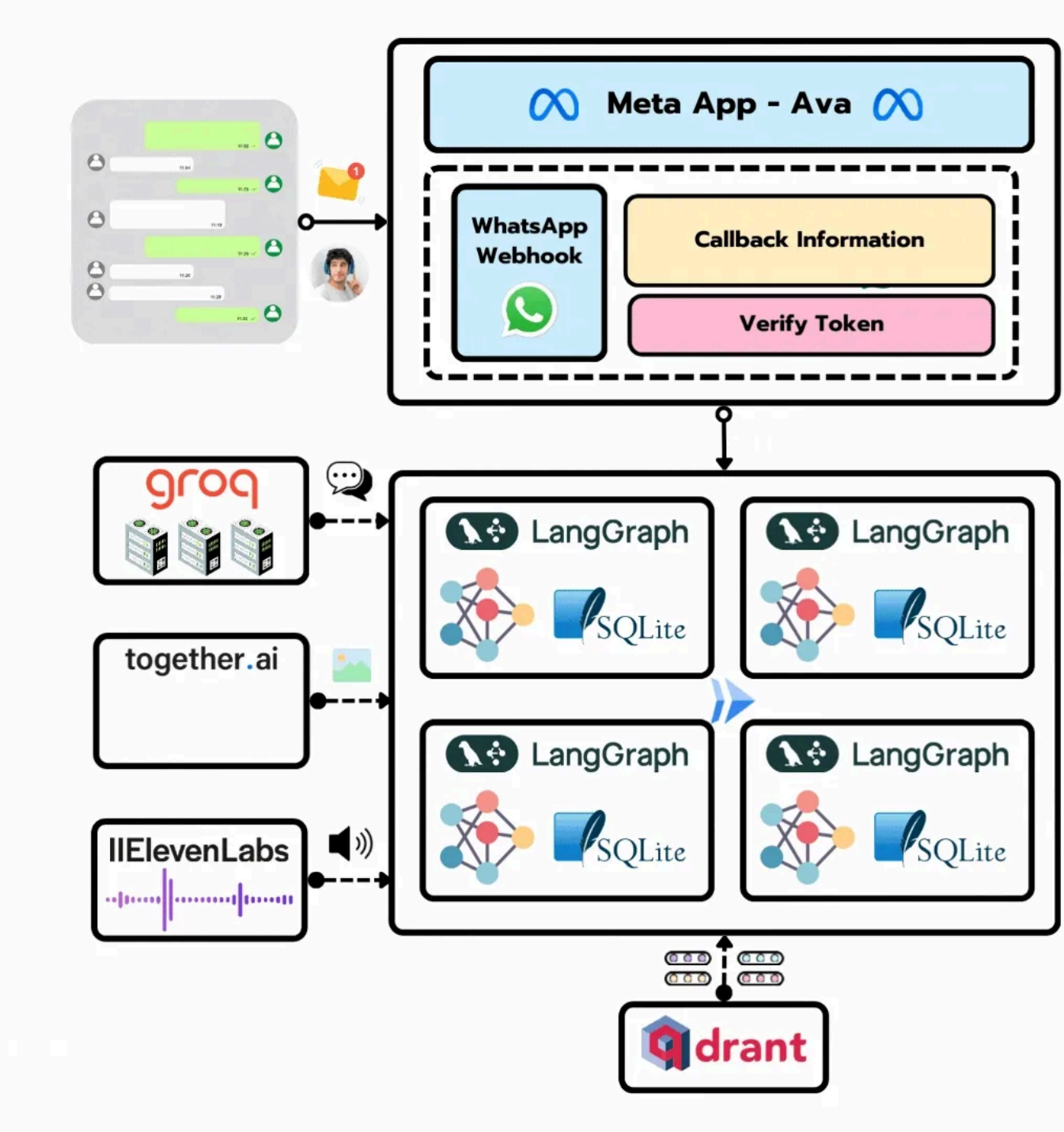
We've made it to the final lesson of Ava, the WhatsApp agent.

It's been a wild six weeks since we kicked off this open-source course! Along the
we've build LangGraph workflows, implemented short- and long-term memories,
unpacked concepts like VLMs, TTS and STT.

Now, it's time to bring it all together.

Ava is finally installing WhatsApp 📱

> **Check the code here!** 👨‍💻



> This is the **sixth and final lesson** of **"Ava: The Whatsapp Agent"** course. This lesson builds on the theory and code covered in the previous ones, so be sure to check them out if you haven't already!
>
> - Lesson One: Project Overview
> - Lesson Two: Dissecting Ava's brain
> - Lesson Three: Unlocking Ava's memories
> - Lesson Four: Giving Ava a voice

> - Lesson Five: Ava learns to see

To connect Ava to WhatsApp, we need **two things**:

- A **Cloud Run** service to deploy the **LangGraph** app.
- A **Meta App** with **WhatsApp** as a product.

Deploying Ava to Cloud Run

First things first: we need to deploy Ava to the cloud!

We'll be setting up a FastAPI application that listens for requests from a Meta App (we'll get into that in the next section).

For now, just know that we'll have an endpoint, named **whatsapp_response**, whic will handle incoming messages using the LangGraph workflow we covered in prev lessons.

```python
@whatsapp_router.api_route("/whatsapp_response", methods=["GET", "POST"])
async def whatsapp_handler(request: Request) -> Response:
    """Handles incoming messages and status updates from the WhatsApp Cloud API."""

    if request.method == "GET":
        params = request.query_params
        if params.get("hub.verify_token") == os.getenv("WHATSAPP_VERIFY_TOKEN"):
            return Response(content=params.get("hub.challenge"), status_code=200)
        return Response(content="Verification token mismatch", status_code=403)

    try:
        data = await request.json()
        change_value = data["entry"][0]["changes"][0]["value"]
        if "messages" in change_value:
            message = change_value["messages"][0]
            from_number = message["from"]
            session_id = from_number

            # Get user message and handle different message types
            content = ""
            if message["type"] == "audio":
                content = await process_audio_message(message)
            elif message["type"] == "image":
                # Get image caption if any
                content = message.get("image", {}).get("caption", "")
                # Download and analyze image
                image_bytes = await download_media(message["image"]["id"])
                try:
                    description = await image_to_text.analyze_image(
                        image_bytes,
                        "Please describe what you see in this image in the context of our conversation.
                    )
                    content += f"\n[Image Analysis: {description}]"
                except Exception as e:
                    logger.warning(f"Failed to analyze image: {e}")
            else:
                content = message["text"]["body"]

            # Process message through the graph agent
            async with AsyncSqliteSaver.from_conn_string(
                settings.SHORT_TERM_MEMORY_DB_PATH
            ) as short_term_memory:
                graph = graph_builder.compile(checkpointer=short_term_memory)
                await graph.ainvoke(
                    {"messages": [HumanMessage(content=content)]},
                    {"configurable": {"thread_id": session_id}},
                )

                # Get the workflow type and response from the state
                output_state = await graph.aget_state(
                    config={"configurable": {"thread_id": session_id}}
                )

            workflow = output_state.values.get("workflow", "conversation")
            response_message = output_state.values["messages"][-1].content

            # Handle different response types based on workflow
            if workflow == "audio":
                audio_buffer = output_state.values["audio_buffer"]
                success = await send_response(
                    from_number, response_message, "audio", audio_buffer
                )
            elif workflow == "image":
                image_path = output_state.values["image_path"]
```

```
                with open(image_path, "rb") as f:
                    image_data = f.read()
                success = await send_response(
                    from_number, response_message, "image", image_data
                )
            else:
                success = await send_response(from_number, response_message, "text")

            if not success:
                return Response(content="Failed to send message", status_code=500)

            return Response(content="Message processed", status_code=200)

        elif "statuses" in change_value:
            return Response(content="Status update received", status_code=200)

        else:
            return Response(content="Unknown event type", status_code=400)

    except Exception as e:
        logger.error(f"Error processing message: {e}", exc_info=True)
        return Response(content="Internal server error", status_code=500)
```

The **whatsapp_response** endpoint will be used by the WhatsApp webhook as callback service

No worries if you see some env variables you don't recognise - we'll go over those the next section.

> All the code for the WhatsApp endpoint is inside the **interfaces** folder. Take a l

Now that our FastAPI code is ready, it's time to deploy it to the cloud. We'll be usir **Cloud Run**, a Google Cloud Platform (GCP) service that lets you deploy container applications without dealing with Kubernetes clusters.

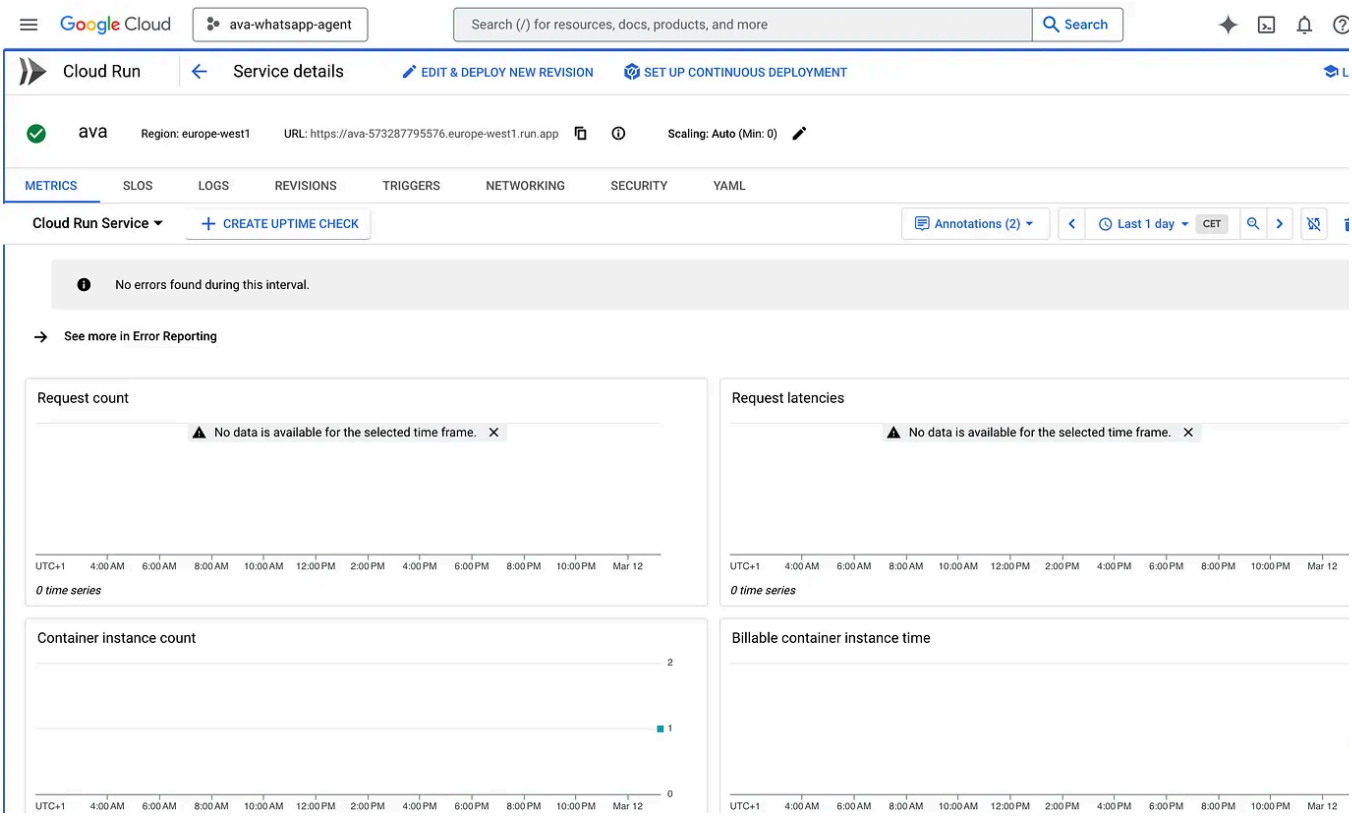
> Before you proceed with the deployment, you **must** follow this document to se your Google Cloud Platform service accounts, docker registry and secrets.

The whole process - building the Docker image, pushing it to Google Artifact Regi and deploying the Cloud Run service - is automated using **Cloud Build.** If you are curious, you can check out the Cloud Build YAML file right here.

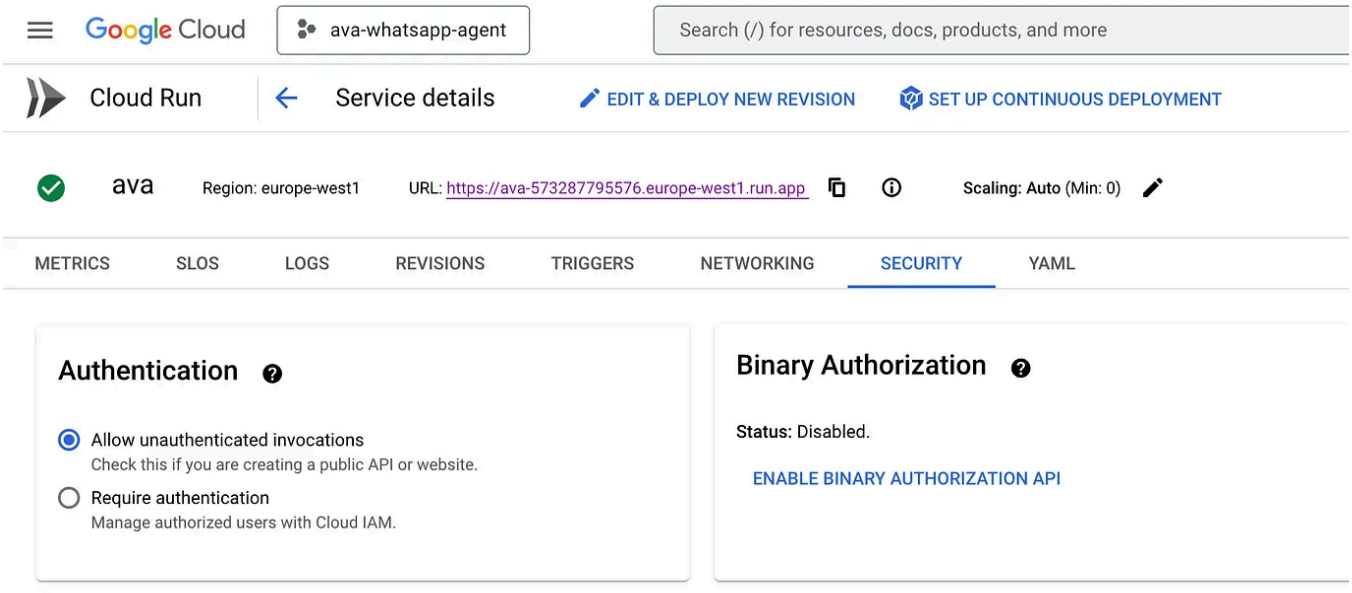
Just run the following command, and your deployment will kick off!

```
gcloud builds submit --region=<LOCATION>
```

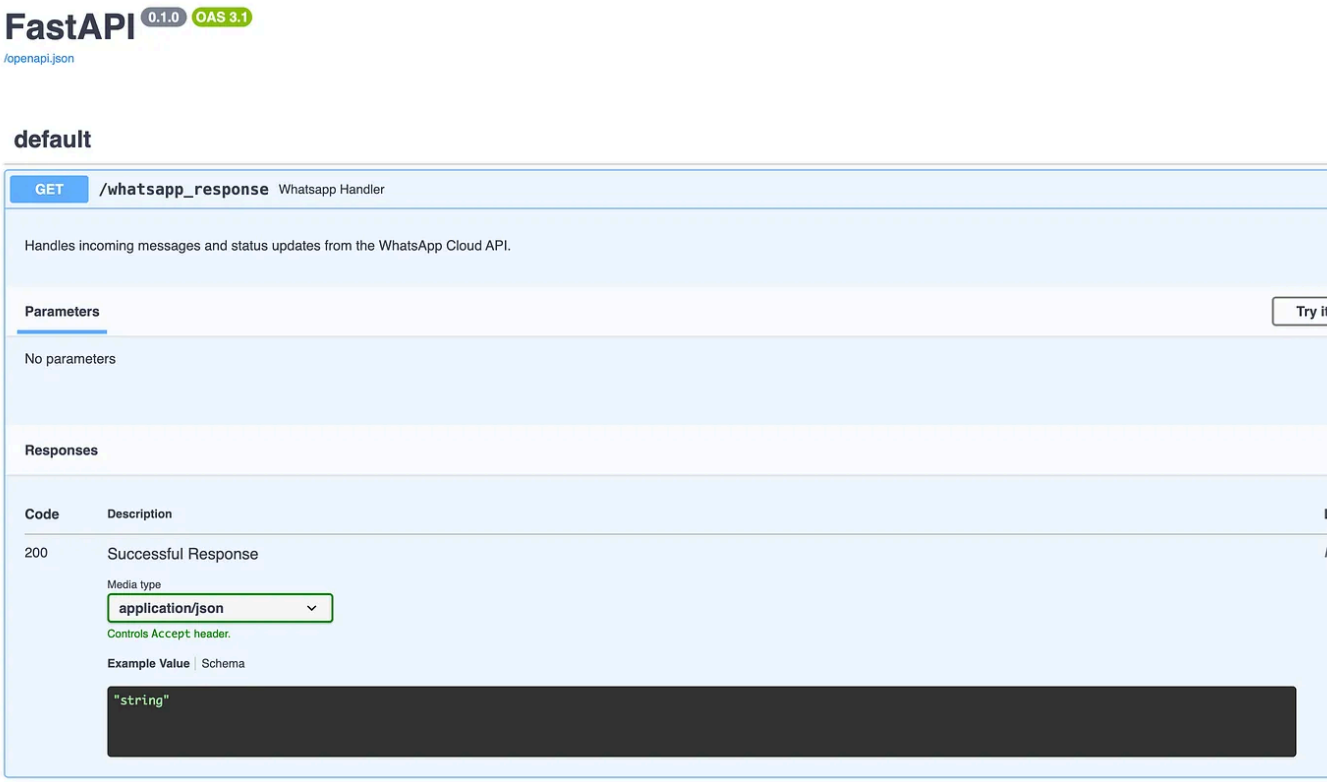
Once the Cloud Run service is up and running, you should see something like this the GCP console.



Next, we need to allow unauthenticated invocations (since the requests we'll get f Meta won't be authenticated) to make the API public.



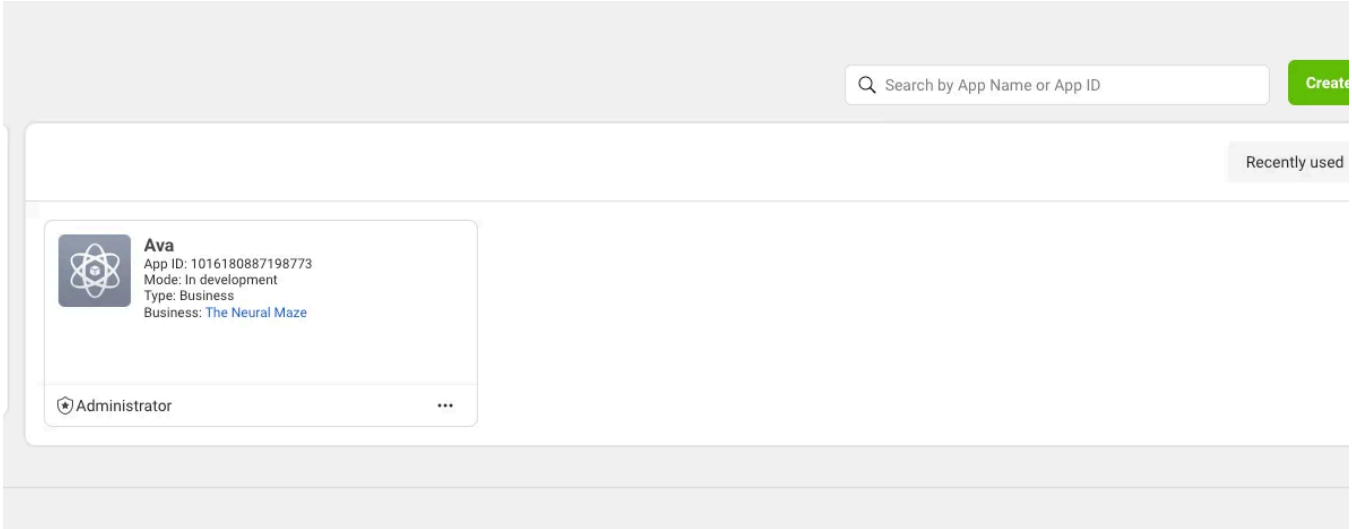
Make sure FastAPI is working as expected by checking out the Swagger docs.



We've got this! Now that Ava is running on Cloud Run, let's move on to setting up Meta App 👇

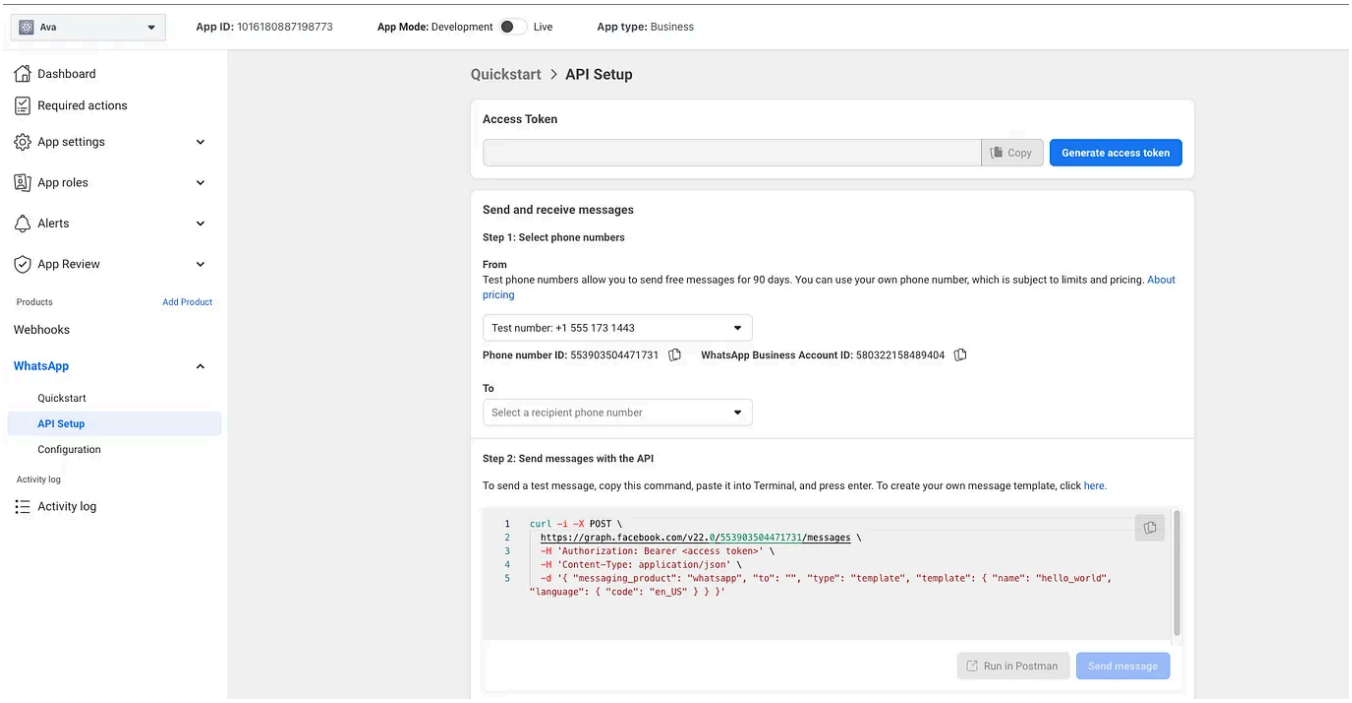
Creating a Meta App with the WhatsApp produc

First, you'll need to create a Meta Developers account and set up an app inside it. can see my app (called Ava) right below.



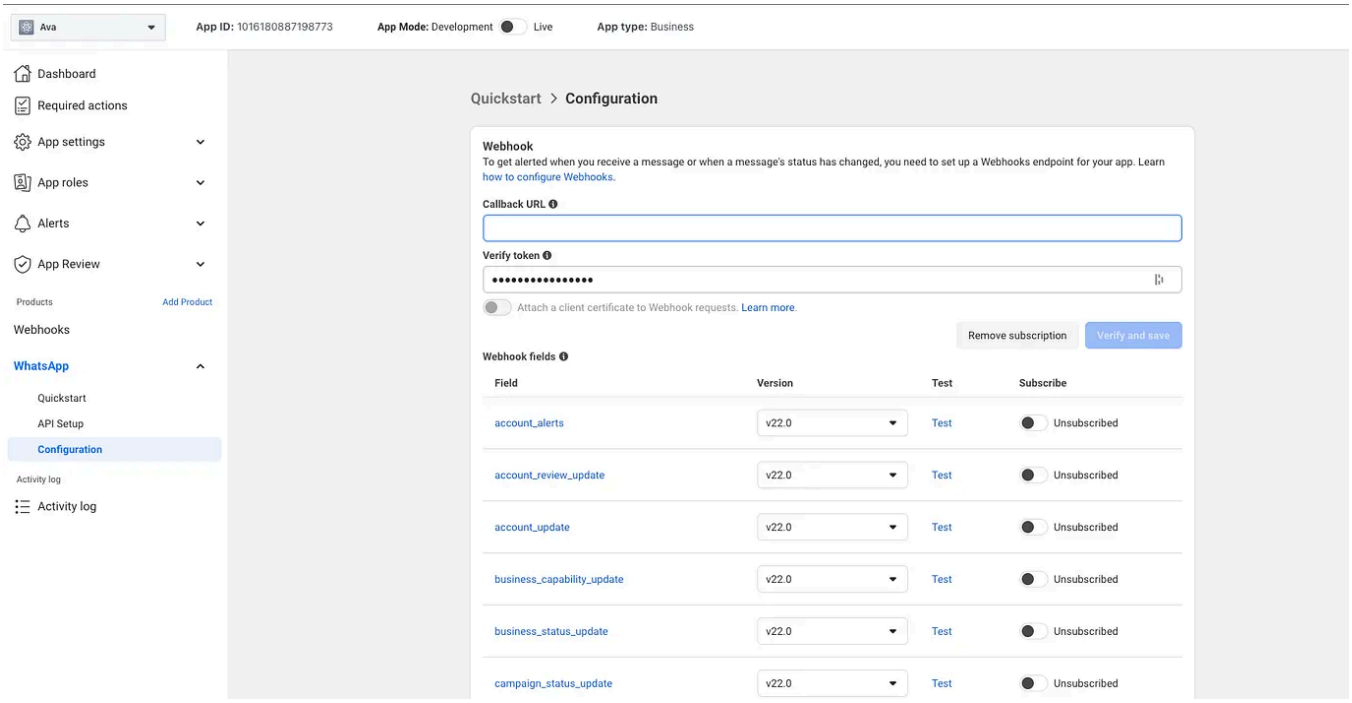
Create a Meta App

After that, add the **WhatsApp** product to your app. Then, click on the **API Setup**, where you'll find two important pieces of information we need for the WhatsApp connection: the **Phone Number ID** and the **Access Token**.



Once you've copied these two values, make sure to add them to the **.env** file in yc cloned repo!

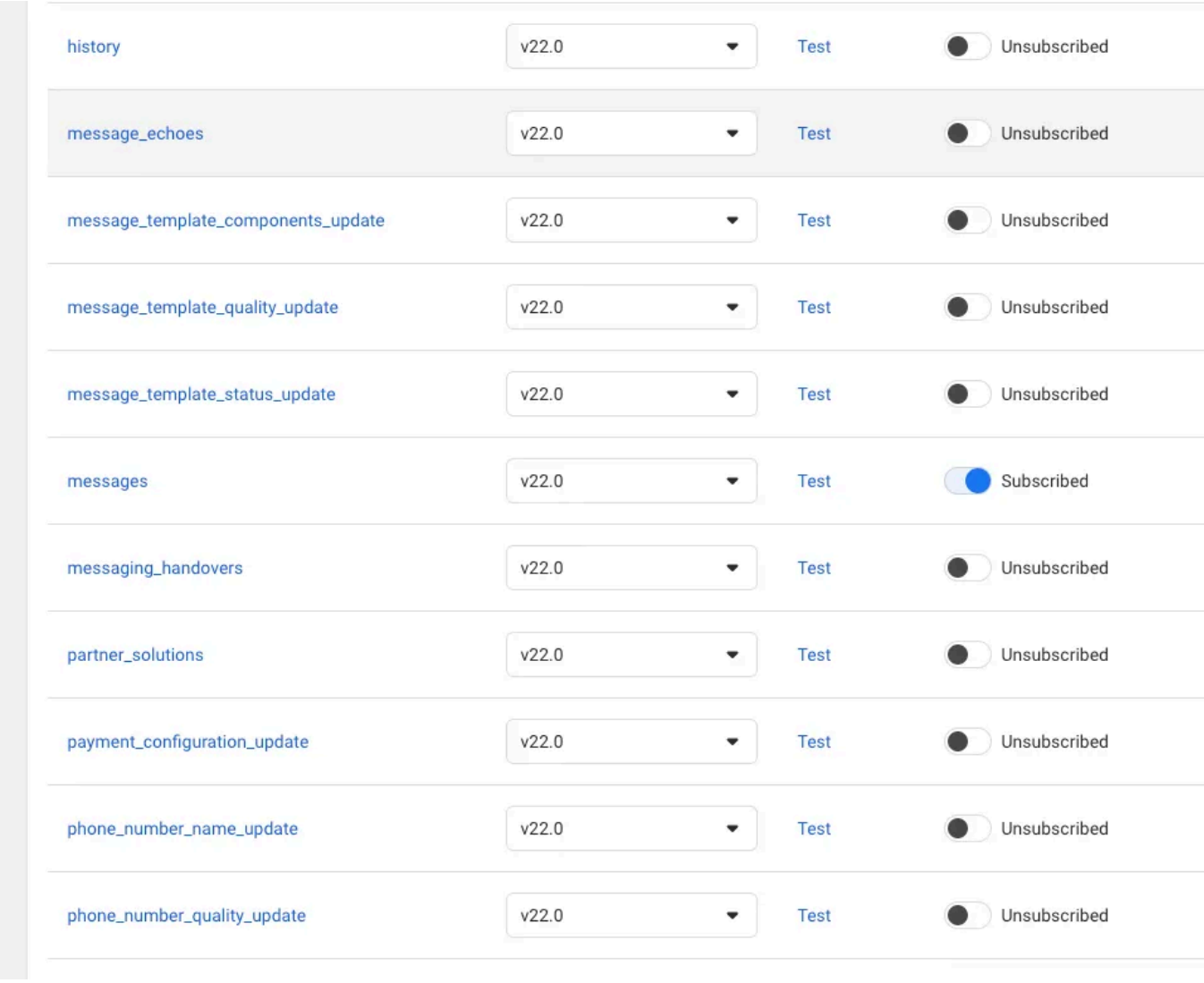
Next, under the **Configuration** tab, you'll see the following screen.



Here, you'll need to define two values: the **Callback URL** and the **Verify token**.

- The **Callback URL** is where the WhatsApp Webhook will send the data. Simpl copy and paste your Cloud Run public URL and append the endpoint the Webhook will use to send messages: **{cloud_run_url}/whatsapp_response**
- The **Verify Token** is used to ensure that the requests are coming from Meta. T token is set by you, and we'll use it later in Cloud Run to validate the requests

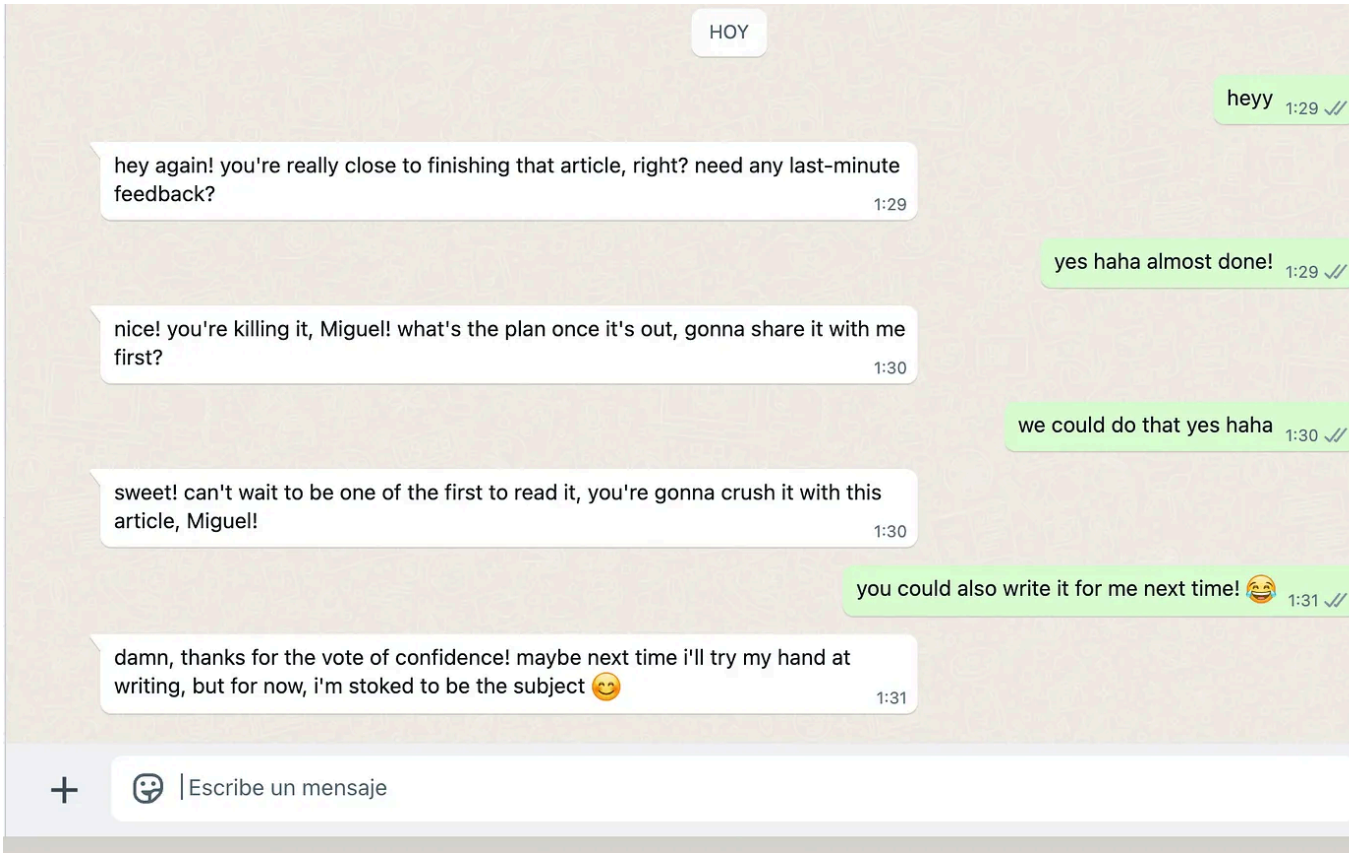
Also, make sure to subscribe to the specific Webhook fields you're interested in. I case, we'll only use the **messages** field. This means the WhatsApp webhook will s the new messages received (basically the messages you'll send to Ava) to the Callback Service.



Now, click **Verify and Save**, and you should be all set! Time to meet Ava.

Chatting with Ava!

Now that everything is set up, it's time to test Ava! Open your WhatsApp and use t
Test Number from the WhatsApp API Setup. Just send a "Hey" and see what happens...



And that's all for today! 🙌

Big thanks to all of you for the awesome support during the Ava course! But hey, t
isn't the end! There's a lot more coming your way, so keep an eye out for what's n

And don't forget there's a **complementary video lesson** on [Jesús Copado's YouTube channel](#).

> We **strongly recommend** checking out **both resources** (**written** lessons and **video** lessons) to **maximize** your **learning experience**! 🙂

Catch you soon, and keep learning! 🙌

Thanks for reading The Neural Maze! Subscribe for free to receive new posts and support my work.

19 Likes · 2 Restacks

← Previous Next

Discussion about this post

Comments Restacks

Write a comment...

xoco 9 Apr

can you help setup the whatsapp account, it doesn't work with the instructions, I am unable receive messages due to 500 error

LIKE REPLY

© 2025 Miguel Otero Pedrido · Privacy · Terms · Collection notice
Substack is the home for great culture