

Taller 1 – Sistema Conversión Cloud

Gina Eveling Posada , Martin Daniel Rincón, Juan Camilo Muñoz, Felipe Serrano

MINE semestre 202410

Universidad de los Andes, Bogotá, Colombia

{g.posadas, md.rincon, jc.munozc12, ff.serrano42}@uniandes.edu.co

Fecha de presentación: abril 09 de 2024

[github](#), [video sustentacion](#)

1) Arquitectura de referencia

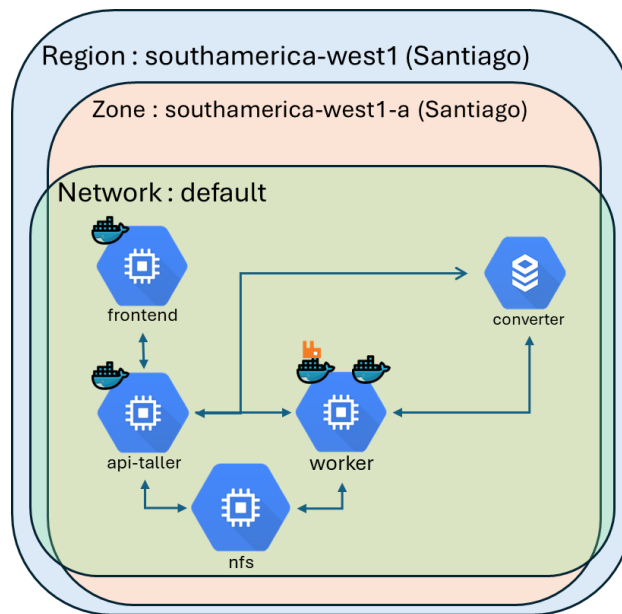


Ilustración 1 Arquitectura general de la aplicación

La aplicación ha sido desarrollada implementando 4 componentes de Docker, un servicio CLOUD SQL, al igual que UN servidor (MV) que sirve de servidor NFS para el almacenamiento de archivos. Estos componentes interactúan en la nube de GCP, dentro de la misma región/zona y red para poder desplegar y cumplir con la funcionalidad requerida para convertir archivos de diferentes (docx, pptx, xlsx, odt) formatos a PDF:

1. Frontend:

- Desarrollado utilizando Streamlit Web API para garantizar una presentación ágil y eficiente para el usuario.
- Facilita la interacción con los servicios del API Rest del Backend a través del protocolo HTTP.

2. Api-taller:

- Implementado en el framework FastApi, proporcionando todas las funcionalidades de negocio.
- Incluye características como registro y login de usuarios mediante JSON Web Tokens (JWT).

- Gestiona la carga y descarga de archivos, así como la gestión de mensajes en la cola de conversión de archivos.

3. Converter (Base de Datos):

- Para esta entrega se está utilizando el servicio SQL ofrecido por la plataforma GCP (CloudSQL), como motor de db se utiliza PostgreSQL para alojar las tablas relacionadas con usuarios y las tareas de conversión de documentos.

4. Worker (Cola de Mensajes rabbitMQ):

- Cada solicitud de conversión genera un mensaje en la cola de rabbitMQ la cual se encuentra sobre su propio contenedor dentro de la maquina denominada Worker

5. Worker (Consumer):

- Programa en python, ejecutado en su propio contenedor, el cual Procesa los mensajes puestos en la cola rabbit para la conversión de archivos de manera asíncrona. Recupera la ruta desde la que debe leer el archivo desde la base de datos y hace llamado al comando en Linux para el libreoffice, y así convertir utilizando este componente gratuito un documento a formato pdf

```
subprocess.call(['soffice', '--headless', '--convert-to', 'pdf', '--outdir', upload_folder, source_file_without_first_slash])
```

- Por último guarda la ruta en la que se almaceno el archivo en formato pdf
- Este contenedor tiene como base una imagen sobre la cual se instaló **Libreoffice lo que lo hace mas escalable, ya que libreoffice hace parte de la imagen y no de la máquina.**

```
RUN apt-get update && apt-get install -y --no-install-recommends \
  libreoffice \
  && rm -rf /var/lib/apt/lists/*
```

Ilustración 2 Comando para instalar libreoffice en la imagen

6. NFS (máquina virtual):

- Máquina virtual que sirve como servidor NFS para que la maquina api y worker puedan intercambiar los archivos. Para la configuración de esta máquina se siguieron los pasos del VIDEO del siguiente enlace <https://jpadillaa.hashnode.dev/tutorial-nfs-network-file-system>


A continuación se colocan imágenes de referencia¹ de la arquitectura desplegada en la nube publica GCP

VM instances							
Filter Enter property name or value							
<input type="checkbox"/> Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP	Connect
<input checked="" type="checkbox"/>	api-taller	southamerica-west1-a			10.194.0.2 (nic0)	34.176.7.107 (nic0)	SSH ▾ ⋮
<input checked="" type="checkbox"/>	frontend	southamerica-west1-a			10.194.0.3 (nic0)	34.176.55.235 (nic0)	SSH ▾ ⋮
<input checked="" type="checkbox"/>	server-nfs	southamerica-west1-a			10.194.0.4 (nic0)	34.176.20.32 (nic0)	SSH ▾ ⋮
<input checked="" type="checkbox"/>	worker	southamerica-west1-a			10.194.0.5 (nic0)	34.176.51.211 (nic0)	SSH ▾ ⋮

Ilustración 3 Lista de máquinas virtuales.


¹ Algunas de las imágenes pueden no coincidir con las IPs del video, ya que se habían acabado los créditos de la cuenta donde se hizo el despliegue


Nombre de maquina	Características
api-taller ; frontend; worker	Tipo de maquina: e2-medium (1-2 vcpu ; 4 Gig RAM; OS debian-12-bookworm-v20240312; SSD 30 GB)
server-nfs	Tipo de maquina: e2-standard-4 (4 vcpu ; 16 Gig RAM; OS ubuntu-2204-jammy-v20240319 ; SSD 250 GB)



SQL

Instances

 CREATE INSTANCE

 MIGRATE DATA

 Filter

Enter property name or value




<input type="checkbox"/>	Instance ID  	Issues	Cloud SQL edition	Type	Public IP address	Private IP address	Instance connection name	High availability
<input type="checkbox"/>	 converter		Enterprise	PostgreSQL 15	34.176.45.253 	10.66.192.15	myfirstproject-41770... 	ENABLE

Ilustración 4 SQL como servicio GCP²

Nombre de instancia	Características
converter	PostgreSQL15; Enterprise; Sandbox (2Vcpus;8GB RAM; Storage 10GB)

- Aspectos transversales para la aplicación:

- Seguridad:

- Se implementan prácticas de seguridad, como la gestión adecuada de tokens JWT y la validación de datos de entrada para prevenir vulnerabilidades.

- Despliegue:

- Se utiliza Docker Compose para la creación de los contenedores de la aplicación, facilitando el despliegue y la escalabilidad (Se elimino el contenedor de Postgress, ya que se esta utilizando la base de datos como un servicio en GCP)
- La infraestructura puede ser fácilmente replicada y gestionada, garantizando consistencia en diferentes entornos.
- No se deben crear variables ni conexiones de “red” a nivel de contenedores, lo que hace más fácil su despliegue, ya que reduce la carga de dependencia de comandos en el contenedor.
- Es importante que al momento de crear los servicios en GCP, se:
 - Creen en la misma zona y región
 - Se modifiquen las reglas de firewall, en la capa de seguridad de la red para abrir los puertos
 - 8501;5001;5432;5672

- Desarrollo y documentación:

- Manejo de versiones de aplicación con repositorio de desarrollo unificado en GITHUB.
- Set de Pruebas de consumo de servicios apoyados con la aplicación de POSTMAN y JMETER

- Escalabilidad:

² Esta imagen puede ser diferente de la del video ya que se habían acabado los créditos en la cuenta donde se hizo el despliegue

- La arquitectura de contenedores permite escalar los distintos componentes de manera independiente, asegurando la capacidad de manejar cargas variables.
- NO se instalaron componentes dentro de las maquinas virtuales, y todo lo necesario esta dentro de la imagen de cada componente, lo que hace que sea más fácil escalar ya que no requiere prerequisites ni actividades manuales para el óptimo funcionamiento del sistema

2) Paso a paso Despliegue en una nube pública (GCP-AWS):

a) Descargar las siguientes imágenes desde docker hub

`docker pull rabbitmq --> el rabbit`

`docker pull fferrano42/fastapi:2203204 --> imagen de fastapi`

`docker pull fferrano42/t1_streamlit:02192024 --> Imagen del streamlit`

`docker pull fferrano42/consumer:24032024 --> Consumidor de la cola`

b) Comandos de docker útiles

`docker rm [id del contenedor]` --> elimina el contenedor

`docker rmi [id de la imagen]` --> elimina la imagen

`-v /home/[nombreusuariogcp]/remote_folder:/remote_folder` → hace la adición de una ruta local a docker para que puedan hacer llamados los programas a una ruta en específico.

c) Pasos para levantar la app por primera vez en la máquina virtual

(1) `docker run --name rabbitmq -p 5672:5672 -p 15672:15672 [id_imagen]`

(2) `docker run --name fast_api -p 5001:5001 -e RABBIT=10.194.0.5 -e DATABASE_URL='postgresql://api:Uniandes2025!@10.66.192.9:5432/convert' -e TOCONVERT=/remote_folder -v /home/[nombreusuariogcp]/remote_folder:/remote_folder [id_imagen]`

(a) En las variables de DATABASE_URL debe ir el valor entre ' ya que se tiene el caracter de ! o de lo contrario Linux lo detecta como un carácter de salida

(b) Se debe obtener la ip privada del servicio de SQL de GCP y configurar de manera correcta

(3) `docker run --name streamlit -p 8501:8501 -e API_URL=http://10.194.0.2:5001 [id_imagen]`

(4) `docker run --name consumer --privileged -e RABBITMQ_URL=10.194.0.5 -e DATABASE='postgresql://api:Uniandes2025!@10.66.192.9:5432/convert' -e TOCONVERT=remote_folder/ -v /home/[nombreusuariogcp]/remote_folder:/remote_folder [id_imagen]`

(a) En las variables de DATABASE_URL debe ir el valor entre ' ya que se tiene el carácter de !

(b) Se debe obtener la ip privada del servicio de SQL de GCP y configurar de manera correcta

(c) Es recomendable que este contenedor sea el Ultimo en iniciar, y después de haber depositado un mensaje en la cola (que de rabbit). Esto debido a que por el momento si no hay mensaje en la cola, el consumidor se apagará.

(d) Para depositar un mensaje en la cola, favor ingresar al portal WEB y dejar cargado un documento

d) Pasos para levanta la app después de haber corrido los contenedores y que la maquina se haya apagado

(1) Obtener el listado de todos los contenedores con `docker ps -a`

(2) Utilizar el comando `docker start [id contenedor]` para iniciar el contenedor

(3) Se recomienda la siguiente secuencia si se desea levantar, en orden de independencia

(4) Levantar la pagina WEB (Streamlit)

(5) Levantar el contenedor de rabbit

(6) Levantar el API (fast_api)

- ### 3) Paso a paso despliegue en visual studio Code:

- 1) Descargar el código fuente del portal de [github](#)
- 2) Para ejecutar el consumer, se debería:
 - i) Ejecutar desde una maquina Linux, que tenga instalado [libreoffice](#)
 - ii) Ejecutar desde una maquina Windows que tenga UNICAMENTE instalado [libreoffice](#) (no recomendado)
- 3) Instalar los módulos correspondientes a cada una de las soluciones, algunos son
 - a. Fast_api
 - b. Streamlit
 - c. Jose (Jwt)
 - d. Pika (gestión de colas de RabbitMq)
 - e. Chardet (gestión en el frontend)

```

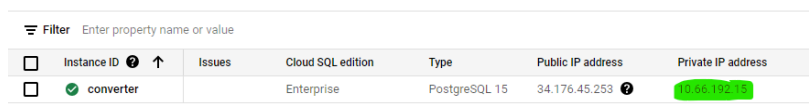
1  # docker-compose.yml
2  #
3  #
4  #
5  #
6  #
7  #
8  #
9  #
10 #
11 #
12 #
13 #
14 #
15 #
16 #
17 #
18 #
19 #
20 #
21 #
22 #
23 #
24 #
25 #
26 #
27 #
28 #
29 #
30 #
31 #
32 #
33 #
34 #
35 #
36 #
37 #
38 #
39 #
40 #
41 #
42 #
43 #
44 #
45 #
46 #
47 #
48 #
49 #
50 #
51 #
52 #
53 #
54 #
55 #
56 #
57 #
58 #
59 #
60 #
61 #
62 #
63 #
64 #
65 #
66 #
67 #
68 #
69 #
70 #
71 #
72 #
73 #
74 #
75 #
76 #
77 #
78 #
79 #
80 #
81 #
82 #
83 #
84 #
85 #
86 #
87 #
88 #
89 #
90 #
91 #
92 #
93 #
94 #
95 #
96 #
97 #
98 #
99 #
100 #
101 #
102 #
103 #
104 #
105 #
106 #
107 #
108 #
109 #
110 #
111 #
112 #
113 #
114 #
115 #
116 #
117 #
118 #
119 #
120 #
121 #
122 #
123 #
124 #
125 #
126 #
127 #
128 #
129 #
130 #
131 #
132 #
133 #
134 #
135 #
136 #
137 #
138 #
139 #
140 #
141 #
142 #
143 #
144 #
145 #
146 #
147 #
148 #
149 #
150 #
151 #
152 #
153 #
154 #
155 #
156 #
157 #
158 #
159 #
160 #
161 #
162 #
163 #
164 #
165 #
166 #
167 #
168 #
169 #
170 #
171 #
172 #
173 #
174 #
175 #
176 #
177 #
178 #
179 #
180 #
181 #
182 #
183 #
184 #
185 #
186 #
187 #
188 #
189 #
190 #
191 #
192 #
193 #
194 #
195 #
196 #
197 #
198 #
199 #
200 #
201 #
202 #
203 #
204 #
205 #
206 #
207 #
208 #
209 #
210 #
211 #
212 #
213 #
214 #
215 #
216 #
217 #
218 #
219 #
220 #
221 #
222 #
223 #
224 #
225 #
226 #
227 #
228 #
229 #
230 #
231 #
232 #
233 #
234 #
235 #
236 #
237 #
238 #
239 #
240 #
241 #
242 #
243 #
244 #
245 #
246 #
247 #
248 #
249 #
250 #
251 #
252 #
253 #
254 #
255 #
256 #
257 #
258 #
259 #
260 #
261 #
262 #
263 #
264 #
265 #
266 #
267 #
268 #
269 #
270 #
271 #
272 #
273 #
274 #
275 #
276 #
277 #
278 #
279 #
280 #
281 #
282 #
283 #
284 #
285 #
286 #
287 #
288 #
289 #
290 #
291 #
292 #
293 #
294 #
295 #
296 #
297 #
298 #
299 #
300 #
301 #
302 #
303 #
304 #
305 #
306 #
307 #
308 #
309 #
310 #
311 #
312 #
313 #
314 #
315 #
316 #
317 #
318 #
319 #
320 #
321 #
322 #
323 #
324 #
325 #
326 #
327 #
328 #
329 #
330 #
331 #
332 #
333 #
334 #
335 #
336 #
337 #
338 #
339 #
340 #
341 #
342 #
343 #
344 #
345 #
346 #
347 #
348 #
349 #
350 #
351 #
352 #
353 #
354 #
355 #
356 #
357 #
358 #
359 #
360 #
361 #
362 #
363 #
364 #
365 #
366 #
367 #
368 #
369 #
370 #
371 #
372 #
373 #
374 #
375 #
376 #
377 #
378 #
379 #
380 #
381 #
382 #
383 #
384 #
385 #
386 #
387 #
388 #
389 #
390 #
391 #
392 #
393 #
394 #
395 #
396 #
397 #
398 #
399 #
400 #
401 #
402 #
403 #
404 #
405 #
406 #
407 #
408 #
409 #
410 #
411 #
412 #
413 #
414 #
415 #
416 #
417 #
418 #
419 #
420 #
421 #
422 #
423 #
424 #
425 #
426 #
427 #
428 #
429 #
430 #
431 #
432 #
433 #
434 #
435 #
436 #
437 #
438 #
439 #
440 #
441 #
442 #
443 #
444 #
445 #
446 #
447 #
448 #
449 #
450 #
451 #
452 #
453 #
454 #
455 #
456 #
457 #
458 #
459 #
460 #
461 #
462 #
463 #
464 #
465 #
466 #
467 #
468 #
469 #
470 #
471 #
472 #
473 #
474 #
475 #
476 #
477 #
478 #
479 #
480 #
481 #
482 #
483 #
484 #
485 #
486 #
487 #
488 #
489 #
490 #
491 #
492 #
493 #
494 #
495 #
496 #
497 #
498 #
499 #
500 #
501 #
502 #
503 #
504 #
505 #
506 #
507 #
508 #
509 #
510 #
511 #
512 #
513 #
514 #
515 #
516 #
517 #
518 #
519 #
520 #
521 #
522 #
523 #
524 #
525 #
526 #
527 #
528 #
529 #
530 #
531 #
532 #
533 #
534 #
535 #
536 #
537 #
538 #
539 #
540 #
541 #
542 #
543 #
544 #
545 #
546 #
547 #
548 #
549 #
550 #
551 #
552 #
553 #
554 #
555 #
556 #
557 #
558 #
559 #
560 #
561 #
562 #
563 #
564 #
565 #
566 #
567 #
568 #
569 #
570 #
571 #
572 #
573 #
574 #
575 #
576 #
577 #
578 #
579 #
580 #
581 #
582 #
583 #
584 #
585 #
586 #
587 #
588 #
589 #
590 #
591 #
592 #
593 #
594 #
595 #
596 #
597 #
598 #
599 #
600 #
601 #
602 #
603 #
604 #
605 #
606 #
607 #
608 #
609 #
610 #
611 #
612 #
613 #
614 #
615 #
616 #
617 #
618 #
619 #
620 #
621 #
622 #
623 #
624 #
625 #
626 #
627 #
628 #
629 #
630 #
631 #
632 #
633 #
634 #
635 #
636 #
637 #
638 #
639 #
640 #
641 #
642 #
643 #
644 #
645 #
646 #
647 #
648 #
649 #
650 #
651 #
652 #
653 #
654 #
655 #
656 #
657 #
658 #
659 #
660 #
661 #
662 #
663 #
664 #
665 #
666 #
667 #
668 #
669 #
670 #
671 #
672 #
673 #
674 #
675 #
676 #
677 #
678 #
679 #
680 #
681 #
682 #
683 #
684 #
685 #
686 #
687 #
688 #
689 #
690 #
691 #
692 #
693 #
694 #
695 #
696 #
697 #
698 #
699 #
700 #
701 #
702 #
703 #
704 #
705 #
706 #
707 #
708 #
709 #
710 #
711 #
712 #
713 #
714 #
715 #
716 #
717 #
718 #
719 #
720 #
721 #
722 #
723 #
724 #
725 #
726 #
727 #
728 #
729 #
730 #
731 #
732 #
733 #
734 #
735 #
736 #
737 #
738 #
739 #
740 #
741 #
742 #
743 #
744 #
745 #
746 #
747 #
748 #
749 #
750 #
751 #
752 #
753 #
754 #
755 #
756 #
757 #
758 #
759 #
760 #
761 #
762 #
763 #
764 #
765 #
766 #
767 #
768 #
769 #
770 #
771 #
772 #
773 #
774 #
775 #
776 #
777 #
778 #
779 #
780 #
781 #
782 #
783 #
784 #
785 #
786 #
787 #
788 #
789 #
790 #
791 #
792 #
793 #
794 #
795 #
796 #
797 #
798 #
799 #
800 #
801 #
802 #
803 #
804 #
805 #
806 #
807 #
808 #
809 #
810 #
811 #
812 #
813 #
814 #
815 #
816 #
817 #
818 #
819 #
820 #
821 #
822 #
823 #
824 #
825 #
826 #
827 #
828 #
829 #
830 #
831 #
832 #
833 #
834 #
835 #
836 #
837 #
838 #
```

- Carpeta API
 - Contiene el proyecto en FAST_API el cual es el API de integración.
 - Para ejecutar el proyecto será necesario estar ubicado sobre el proyecto y en una terminal escribir el comando **uvicorn app:app --port 5001 --reload**
- Carpeta WEB
 - Contiene el proyecto en streamlit el cual es el FRONTEND
 - Para ejecutar el proyecto será necesario estar ubicado sobre el proyecto y en una terminal escribir el comando **streamlit run Login.py**
- Carpeta Rabbit
 - Contiene el proyecto en (consola) python el cual es el consumer de mensaies de rabbit

- Su función es leer el mensaje de la cola, que contiene el id del documento. Luego buscar ese registro en la base de datos postgres, obtener del campo **source_file** que es la ruta compartida donde está el archivo físico. Posterior, ejecuta el siguiente llamado al proceso de libreoffice **subprocess.call(['soffice', '--headless', '--convert-to', 'pdf', '--outdir', 'upload_folder', 'source_file_without_first_slash'])** después de ejecutar el comando, registra en la base de datos, en la columna **pdf_file**, la ruta donde se almaceno el archivo convertido
- Para ejecutar el proyecto será necesario estar ubicado sobre la carpeta y ejecutar la clase **converter.py**
 - **Las otras clases dentro del proyectos son referencia a futuras mejoras (ejercicios) que se deseaban implementar, pero por motivo de tiempo no fue posible eliminarlas**
- Recomendaciones para construir las imágenes desde visual studio:
 - Existe un archivo llamado **docker-compose.yml**, el cual es el compose de todas las imágenes dockerFile de cada proyecto.
 - En este docker-compose, no está la imagen de Rabbit, la cual debe ser descargada y ejecutada por aparte.
 - `docker pull rabbitmq`
 - **`docker run --name rabbitmq -p 5672:5672 -p 15672:15672 [El id de la imagen]`**
 - Para que se genere automáticamente las imágenes, desde visual studio se puede utilizar el comando **docker-compose build**. y este comando construirá las imágenes de
 - Api (Fast_Api)
 - Consumer (Python)
 - Frontend (Streamlit)
 - En caso de ejecutar el comando docker-compose up, pueden presentar fallos si las variables de entorno no se han configurado correctamente en la maquina ó si no están asociadas los contenedores a la misma red. Para esta etapa del taller los contenedores se iban a desplegar en múltiples maquinas, se elimino el uso de una network en común y se recomienda que se mantenga este esquema también en desarrollo.
 - Adicionalmente, la imagen de postgres no se genera de manera automática con el dockerCompose, ya que se desea hacer la prueba directamente conectando el sistema a servicio googleSQL o AWS RDS. Sin embargo se dejó dentro del código fuente una carpeta de postgres (con un dockerfile) por si más adelante los créditos se agotan y se debe utilizar una imagen nuevamente.

4) Limitaciones de la solución

1. Aún no se ha implementado balanceadores de carga para un escalamiento horizontal de la aplicación sin tener que configurar todos los contenedores. Por lo que uno de los límites es la capacidad máxima de peticiones HTTP que acepta cada una de las maquinas, al igual que su proceso de escalabilidad (con la arquitectura actual se debe realizar muchos procesos manuales)
2. Una vez desplegada la solución en las maquina virtuales de GCP al igual que la base de datos SQL de GCP, se recomienda:
 1. Apagar las máquinas virtuales
 2. Eliminar la base de datos de GCP
 3. Esto con el fin de no consumir los créditos, ya que incluso si el servicio de SQL esta detenido, este sigue consumiendo créditos.
3. Por el momento, las maquinas virtuales al igual que el servicio SQL en GCP, no están en redundancias de zonas, y todas se encuentran en la misma zona southamerica-west1-a (esto con el fin de facilitar la comunicación entre componentes). Esto implica que si existe un fallo en esa zona, TODA la aplicación presentaría una falla
4. A pesar de haber habilitado los puertos https y realizar la validación de CORS, las peticiones al API en GCP únicamente se pueden realizar utilizando el protocolo http.
5. Al contar con una MAQUINA VIRTUAL como servidor NFS, si este servidor no se encuentra activo, la funcionalidad principal (conversión) de todo el sistema fallaría, motivo por el cual la configuración y administración de esta maquina virtual es vital para el correcto funcionamiento del sistema
 1. Este es una limitante crítica y de pronto para futuras iteraciones se podía remplazar este servidor, por un servicio como Filestore (en el caso de GCP)
6. Al momento de eliminar la base de datos de GCP, la ip que tienen los contenedores del API y Consumer a la que hacen referencia al servicio SQL queda obsoleta (cuando se crea una nueva instancia se asigna otra IP). Esto hace que cada vez que se cree una nueva instancia de base de datos, se deba:
 1. Eliminar los contenedores que están corriendo
 2. Volver a crear los contenedores con sus correspondientes comando pero modificando la ip PRIVADA de la base de datos.³



Filter Enter property name or value						
<input type="checkbox"/>	Instance ID ⓘ ↑	Issues	Cloud SQL edition	Type	Public IP address	Private IP address
<input type="checkbox"/>	✓ converter		Enterprise	PostgreSQL 15	34.176.45.253 ⓘ	10.86.192.14

Ilustración 6 IP privada de la base de datos

³ También es recomendable habilitar la conexión de la base de datos desde fuera, ya que en algunas ocasiones GCP crea la ip privada dentro de otra red, que no están las maquinas y es mas complejo la conectividad internamente.

5) Conclusiones

1. El uso de contenedores para el despliegue de la solución reduce la carga manual de configuración de componentes uno a uno, además de permitir contar con una arquitectura de solución desacoplada y escalable eficiente ya que solo se debe crecer en el componente necesario (front, back, consumer).
2. La curva de aprendizaje en los múltiples servicios dificulta el tiempo de despliegue final en la nube pública. Es indispensable tener un conocimiento claro de la arquitectura “de red” de los distintos componentes para que la interconexión entre distintas máquinas no se vuelva un inconveniente mayor.
3. Se debe conocer los distintos modelos de cobro que tienen las nubes públicas, ya que dependiendo del servicio y de la nube pública se pueden utilizar estrategias para optimizar el costo total de la solución.
4. A medida que la arquitectura de la solución se va complejizando con máquinas/servicios de nube, el ambiente de desarrollo también se va complejizando para poder hacer una transición sencilla entre desarrollo y despliegue. Es muy importante tener esto definido desde el inicio ya que puede demorar los tiempos de inicio de desarrollo al igual que el despliegue final.