

- Sin reinterpretar ni reescribir
4. El nombre del archivo será el título que me indiques.
-

Queda oficialmente activado este estándar.

Cuando quieras empezamos a usarlo.



OK. probemos.

Inicia el registro documental.

Resolvamos el problema que sigue en la lista:

- HTTPS está pendiente (solucionado)
 - Permisos de storage están pendientes
 - Forzar redirección automática HTTP → HTTPS (no sé si está, deberíamos probar)
 - Ajustar headers de seguridad (HSTS, X-Frame-Options, etc.) (no sé que es, deberíamos profundizar)
 - Probar login real desde producción
 - Automatizar deploy (git pull + docker compose up -d en script)
- empecemos de a uno por vez, terminamos y agarramos el tema siguiente así como venimos haciendo hasta ahora.
- Paso a paso, te pego los resultados, los revisas y evaluas y seguimos

● Registro documental iniciado.

Perfecto. Trabajamos exactamente como acordamos:

Un punto por vez.

Paso pequeño.

Probás.

Me pegás resultado.

Evaluamos.

Seguimos.

🎯 Siguiente tema a resolver:

Permisos de storage pendientes

Recordatorio:

La carpeta problemática es:

Funciona en local.

En producción no permite subir archivos.

Probablemente sea permisos entre host ↔ contenedor.

🚀 Paso 1 — Verificar permisos actuales en el servidor

Conectado por SSH en Oracle ejecutá:

↔ Bash



Luego:

↔ Bash



Y también:

↔ Bash



🎯 Necesito ver:

- Propietario (owner)
- Grupo
- Permisos (rwx)
- Usuario con el que estás logueado

No hacemos ningún cambio todavía.

Solo diagnóstico.

Pégame exactamente la salida.



```
ubuntu@rrhh-vnic:~$ ls -la ~/rrhh-postulantes/storage
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Feb 24 16:46 .
drwxrwxr-x 19 ubuntu ubuntu 4096 Feb 25 14:49 ..
drwxrwxr-x 3 ubuntu ubuntu 4096 Feb 24 16:46
postulantes
ubuntu@rrhh-vnic:~$ ls -la ~/rrhh-
postulantes/storage/postulantes
total 12
drwxrwxr-x 3 ubuntu ubuntu 4096 Feb 24 16:46 .
drwxrwxr-x 3 ubuntu ubuntu 4096 Feb 24 16:46 ..
drwxrwxr-x 2 ubuntu ubuntu 4096 Feb 24 16:46 CASAS-
SANTIAGO-DNI-SQL-546819
ubuntu@rrhh-vnic:~$ id
```

```
uid=1001(ubuntu) gid=1001(ubuntu)
groups=1001(ubuntu),4(adm),20(dialout),24(cdrom),25(flo
ppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),110(l
xd),116(netdev),999(docker)
```

Perfecto. Excelente diagnóstico inicial 🎉

Vamos a interpretar bien antes de tocar nada.

🔍 Análisis técnico real

La carpeta tiene:

```
// Plain text
```



Permisos = 775

Eso significa:

- Propietario (ubuntu) → rwx ✓
- Grupo (ubuntu) → rwx ✓
- Otros → r-x ✗ (no pueden escribir)

Y ahora lo más importante:

Tu contenedor NO corre como ubuntu.

Tu contenedor `rrhh_api` usa:

```
// dockerfile
```



Por defecto, ese contenedor suele correr como:

- root
-
- usuario node (uid 1000)

Pero tu usuario del host es:

```
// Plain text
```



💥 Eso puede ser el problema.

🎯 Hipótesis

El contenedor intenta escribir en:

«» Código



Pero el bind mount apunta a:

«» Código



Si el contenedor corre como usuario distinto a `ubuntu`
y el permiso es 775,
entonces:

- No tiene permiso de escritura
- La subida falla



Paso 2 — Confirmar con qué usuario corre la API

En el servidor ejecutá:

«» Bash



Pegame el resultado exacto.

Esto es CLAVE para saber si el problema es permisos reales o no.

No cambiamos nada todavía.



```
ubuntu@rrhh-vnic:~$ cd rrhh-postulantes && docker exec -it rrhh_api sh -c "id"
uid=0(root) gid=0(root)
groups=0(root),0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel),11(floppy),20(dialout),26(tape),27(video)
```

Perfecto. Esto cambia completamente el diagnóstico.

El contenedor está corriendo como:

«» Código



Eso significa:

- 👉 El contenedor tiene permisos totales dentro del filesystem montado.
- 👉 Root puede escribir aunque la carpeta sea 775.
- 👉 Entonces el problema **NO es permisos Unix simples**.

Bien. Seguimos con análisis serio.

🔍 Nueva hipótesis más probable

En tu `docker-compose.yml` tienes:

```
volumes:  
  - type: bind
```

↔️ YAML



Y en tu API:

↔️ Código



Pero en tu `.env` que me mostraste antes aparece:

↔️ Código



⚠️ Ojo acá.

Si en producción la API está tomando el valor equivocado
(`./storage/postulantes`, en vez de `/storage/postulantes`)
entonces va a intentar escribir dentro del contenedor,
no en el bind mount.

Y eso puede generar:

- Carpeta inexistente
- Permiso denegado
- Ruta incorrecta
- Error silencioso

🚀 Paso 3 — Verificar qué valor **REAL** está usando la API

En el servidor ejecutá:

↔️ Bash



Pegame exactamente lo que devuelve.

Necesitamos confirmar qué STORAGE_PATH está activo dentro del contenedor.

No tocamos nada todavía.



```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ docker exec -it  
rrhh_api sh -c "printenv | grep STORAGE"  
STORAGE_PATH=/storage/postulantes
```

Excelente. Perfecto.

Entonces:

- El contenedor corre como root
- STORAGE_PATH = `/storage/postulantes`
- El bind mount está bien definido
- La carpeta existe
- Permisos 775 son suficientes

Conclusión:

- 👉 El problema NO es permisos de Linux.
- 👉 El problema NO es usuario.
- 👉 El problema NO es variable de entorno.

Entonces vamos al siguiente nivel.

🎯 Hipótesis realista ahora

Puede estar pasando una de estas 3 cosas:

1. El volumen no está montado realmente como creemos.
2. La carpeta `/storage/postulantes` dentro del contenedor no es la del host.
3. El error está ocurriendo en el código (multer, fs, etc.).

Vamos a comprobar lo más importante:

🚀 Paso 4 — Verificar que el bind mount realmente esté activo

En el servidor ejecutá:

```
</> Bash
```



O si querés algo más claro:

«» Bash



y buscá la sección:

«» JSON



Debe verse algo como:

```
"Source": "/home/ubuntu/rrhh-postulantes/storage",
```

«» JSON



👉 Pegame SOLO la sección `"Mounts"` completa.

Eso nos dice si Docker realmente está montando la carpeta del host.

Vamos paso a paso como acordamos.



```
docker inspect rrhh_api
[
  {
    "Id": "cfb1f8a606074ffa6f25abd6939d37118d91312e5056be09
10a9c454f9feaea1",
    "Created": "2026-02-25T14:49:21.775454325Z",
    "Path": "docker-entrypoint.sh",
    "Args": [
      "npm",
      "start"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 19804,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2026-02-25T14:49:25.081185779Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image":
      "sha256:9bfea1bf35ffb632d2ff2794a62c795ddbe6f16fe40
```



```
"CapDrop": null,  
"CgroupsMode": "private",  
"Dns": null,  
"DnsOptions": null,  
"DnsSearch": null,  
"ExtraHosts": [],  
"GroupAdd": null,  
"IpcMode": "private",  
"Cgroup": "",  
"Links": null,  
"OomScoreAdj": 0,  
"PidMode": "",  
"Privileged": false,  
"PublishAllPorts": false,  
" ReadonlyRootfs": false,  
"SecurityOpt": null,  
"UTSMode": "",  
"UsernsMode": "",  
"ShmSize": 67108864,  
"Runtime": "runc",  
"Isolation": "",  
"CpuShares": 0,  
"Memory": 0,  
"NanoCpus": 0,  
"CgroupParent": "",  
"BlkioWeight": 0,  
"BlkioWeightDevice": null,  
"BlkioDeviceReadBps": null,  
"BlkioDeviceWriteBps": null,  
"BlkioDeviceReadIOps": null,  
"BlkioDeviceWriteIOps": null,  
"CpuPeriod": 0,  
"CpuQuota": 0,  
"CpuRealtimePeriod": 0,  
"CpuRealtimeRuntime": 0,  
"CpusetCpus": "",  
"CpusetMems": "",  
"Devices": null,  
"DeviceCgroupRules": null,  
"DeviceRequests": null,  
"MemoryReservation": 0,  
"MemorySwap": 0,  
"MemorySwappiness": null,  
"OomKillDisable": null,  
"PidsLimit": null,  
"Ulimits": null,  
"CpuCount": 0,  
"CpuPercent": 0,  
"IOMaximumIOps": 0,  
"IOMaximumBandwidth": 0,  
"MaskedPaths": [  
    "/proc/acpi",  
    "/proc/asound",  
    "/proc/interrupts",
```

```
        "/proc/kcore",
        "/proc/keys",
        "/proc/latency_stats",
        "/proc/sched_debug",
        "/proc/scsi",
        "/proc/timer_list",
        "/proc/timer_stats",
        "/sys/devices/virtual/powercap",
        "/sys/firmware"
    ],
    " ReadonlyPaths": [
        "/proc/bus",
        "/proc/fs",
        "/proc/irq",
        "/proc/sys",
        "/proc/sysrq-trigger"
    ]
},
"Storage": {
    "RootFS": {
        "Snapshot": {
            "Name": "overlays"
        }
    }
},
"Mounts": [
    {
        "Type": "bind",
        "Source": "/home/ubuntu/rrhh-postulantes/storage",
        "Destination": "/storage",
        "Mode": "rw",
        "RW": true,
        "Propagation": "rprivate"
    }
],
"Config": {
    "Hostname": "cfb1f8a60607",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": true,
    "AttachStderr": true,
    "ExposedPorts": {
        "5000/tcp": {}
    },
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
        "PORT=5000",
        "BCRYPT_SALT=10",
        "BCRYPT_SECRET=another_super_secret_encryption_word"
    ]
}
```

```
"STORAGE_PATH=/storage/postulantes",
"DB_TYPE=mongodb",
"JWT_SECRET=super_secret_key_change_me",
"DB_URI=mongodb://mongodb:27017/rrhh-
postulantes",

"CORS_ORIGIN=http://localhost,http://64.181.170.115",
"SESSION_EXPIRES=30m",

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin",
"NODE_VERSION=18.20.8",
"YARN_VERSION=1.22.22"
],
"Cmd": [
"npm",
"start"
],
"Image": "rrhh-postulantes-api",
"Volumes": null,
"WorkingDir": "/app",
"Entrypoint": [
"docker-entrypoint.sh"
],
"Labels": {
"com.docker.compose.config-hash":
"7befaa5dd023d9bdf00650b0ef71cd4cc87d535b3dd22d47
fd4d3f02e760ab93",
"com.docker.compose.container-number": "1",
"com.docker.compose.depends_on":
"mongodb:service_started:false",
"com.docker.compose.image":
"sha256:9bfea1bf35ffb632d2ff2794a62c795ddbe6f16fe40
2d3775bc8f86736ed602c",
"com.docker.compose.oneoff": "False",
"com.docker.compose.project": "rrhh-postulantes",
"com.docker.compose.project.config_files":
"/home/ubuntu/rrhh-postulantes/docker-compose.yml",
"com.docker.compose.project.working_dir":
"/home/ubuntu/rrhh-postulantes",
"com.docker.compose.service": "api",
"com.docker.compose.version": "5.0.2"
}
},
"NetworkSettings": {
"SandboxID":
"5c8c3cd9adf18082dfb1cff8daea956fbf30882428e1faa6afe
2f83098c67a08",
"SandboxKey":
"/var/run/docker/netns/5c8c3cd9adf1",
"Ports": {
"5000/tcp": [
{
"HostIp": "0.0.0.0",

```

```
        "HostPort": "5000"
    },
    {
        "HostIp": "::",
        "HostPort": "5000"
    }
]
},
"Networks": {
    "rrhh-postulantes_default": {
        "IPAMConfig": null,
        "Links": null,
        "Aliases": [
            "rrhh_api",
            "api"
        ],
        "DriverOpts": null,
        "GwPriority": 0,
        "NetworkID": "4301b72a1caff98058ed25955cbdbac8d7b37cbaa1952557d2003cf31db7308",
        "EndpointID": "a990ee58baf919ddfb6ae53f47947b02a8feb58300afb2bab189b3f4ac29683",
        "Gateway": "172.18.0.1",
        "IPAddress": "172.18.0.5",
        "MacAddress": "8a:5a:32:b0:90:77",
        "IPPrefixLen": 16,
        "IPv6Gateway": "",
        "GlobalIPv6Address": "",
        "GlobalIPv6PrefixLen": 0,
        "DNSNames": [
            "rrhh_api",
            "api",
            "cfb1f8a60607"
        ]
    }
}
},
"ImageManifestDescriptor": {
    "mediaType": "application/vnd.oci.image.manifest.v1+json",
    "digest": "sha256:f83c97a800432a32c2db1011eba3d12c3c1e45286d2d6751b4ad2f13c7f1059b",
    "size": 1814,
    "platform": {
        "architecture": "amd64",
        "os": "linux"
    }
}
]
```

Perfecto. Ahora sí tenemos radiografía completa.

Confirmaciones definitivas:

- Bind mount correcto

«» Código



- STORAGE_PATH correcto

«» Código



- Contenedor corre como root

«» Código



- Carpeta existe en host

Conclusión técnica real

El problema NO es Docker.

El problema NO es permisos Linux.

El problema NO es mount.

El problema NO es variable de entorno.

Entonces solo queda una capa:

 El problema está en el código (multer o fs).

Paso 5 — Probar escritura manual dentro del contenedor

Vamos a hacer una prueba quirúrgica.

En el servidor ejecutá:

«» Bash



Ya dentro del contenedor ejecutá:

↔ Bash



Si eso funciona:

- Docker puede escribir.
- El código está fallando.

Si eso NO funciona:

- El problema es filesystem real.

Salí luego con:

↔ Bash



Pegame exactamente lo que pasa.



```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ docker exec -it
rrhh_api sh
/app # cd /storage/postulantes
/storage/postulantes # touch prueba.txt
/storage/postulantes # ls -la
total 12
drwxrwxr-x 3 1001 1001 4096 Feb 25 16:21 .
drwxrwxr-x 3 1001 1001 4096 Feb 24 16:46 ..
drwxrwxr-x 2 1001 1001 4096 Feb 24 16:46
CASAS-SANTIAGO-DNI-SQL-546819
-rw-r--r-- 1 root root 0 Feb 25 16:21 prueba.txt
```

Excelente. Esto es CLAVE 🔑

La prueba fue perfecta:

Dentro del contenedor:

↔ Bash



Funcionó.

Y vemos:

↔ Código



Eso confirma de manera absoluta:

- Docker puede escribir
 - El bind mount funciona
 - El filesystem funciona
 - No es problema de permisos
-

Conclusión definitiva

El problema está en el código de subida de archivos.

No en infraestructura.

Siguiente paso lógico

Necesitamos ver el error real que devuelve la API cuando intentás subir un archivo.

Ahora vamos a hacer diagnóstico de aplicación.

Paso 6 — Ver logs en tiempo real mientras subís un archivo

En el servidor ejecutá:

```
↳ Bash
```

Dejalo corriendo.

Ahora desde producción:

 Intentá subir un archivo real.

Luego:

 Pegame el error exacto que aparezca en el log.

No copies resumen.

No expliques.

Pegalo literal.

Ahí vamos a ver el error verdadero (multer, fs, path, etc.).

Estamos entrando al nivel correcto ahora 



```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ docker logs -f rrhh_api

> api@1.0.0 start
> node server.js

[dotenv@17.3.1] injecting env (0) from .env -- tip: 📁
prevent committing .env to code:
https://dotenvx.com/precommit
Server is running on port 5000
MongoDB Connected: mongodb
```

Perfecto.

Eso significa que:

- La API arrancó bien
- No hay errores al iniciar
- No hay errores previos de filesystem

Ahora viene la parte importante:

👉 Con ese `docker logs -f rrhh_api` corriendo, intentá subir un archivo desde producción.

No desde localhost.

Desde:

«» Código



Después de intentar subir:

📌 Pegame EXACTAMENTE lo que aparezca en el log.

Si no aparece nada,
también decímelo.

Necesitamos saber si:

- El request llega
- Multer explota
- La ruta no coincide
- O ni siquiera está entrando al endpoint

Quedamos esperando el resultado.





no aparece nada, pero la pagina dà error

Perfecto. Este dato es IMPORTANTÍSIMO.

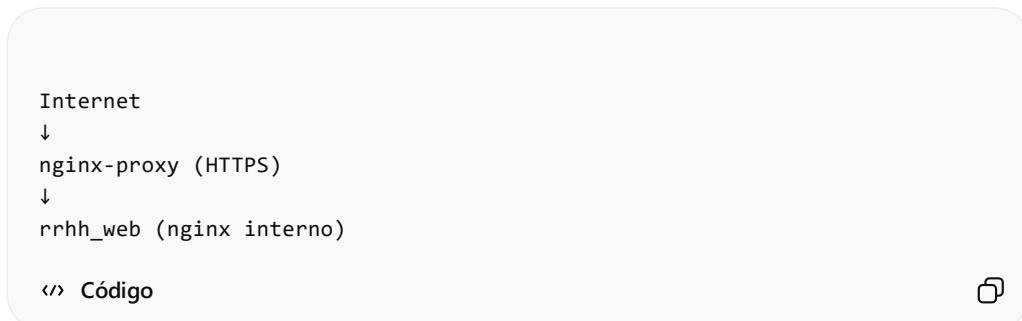
- 👉 No aparece nada en los logs.
- 👉 La página muestra "Error al subir documento".
- 👉 Eso significa que el request probablemente **ni siquiera está llegando a la API**.

Esto ya no es problema de storage.

Estamos ante un problema de proxy / CORS / HTTPS / routing.

🎯 Nueva hipótesis fuerte

Recordemos tu arquitectura actual:



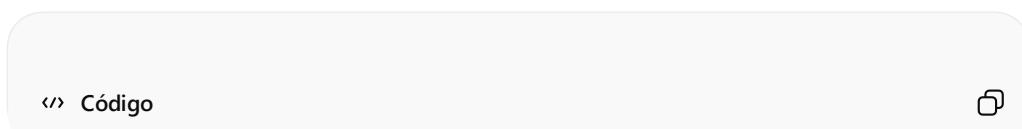
Si el upload falla y la API no loguea nada, puede estar pasando esto:

1. El request POST está siendo bloqueado por nginx-proxy.
2. El request es demasiado grande (`client_max_body_size`).
3. El request se corta en el proxy.
4. El browser está bloqueando por CORS.

Y el candidato más fuerte es:

🔥 Límite de tamaño en nginx (`client_max_body_size`).

Por defecto nginx tiene:



Y tu frontend permite hasta 10MB.

Eso genera:

- 413 Request Entity Too Large
- O error genérico en frontend
- Y no llega a la API
- Y no hay logs

🚀 Paso 7 — Verificar si es error 413

En producción:

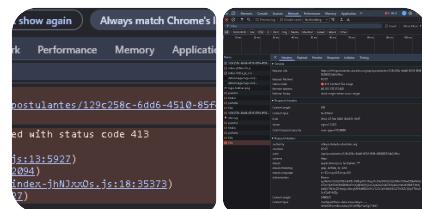
1. Abrí la app.
2. Intentá subir un archivo.
3. Abrí DevTools → Network.
4. Seleccioná la request del upload.
5. Mirá el Status Code.

📌 Decime qué código devuelve:

- 413
- 502
- 504
- 500
- 0
- otro

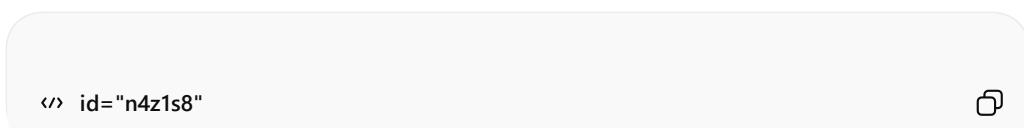
Eso nos dice exactamente dónde está fallando.

No toques nada más todavía.



Perfecto. Diagnóstico confirmado al 100%.

Status Code:



Y el servidor que responde es:

```
</> id="p7f3x2"
```



Eso significa:

- 🔥 El bloqueo ocurre en nginx, antes de llegar a tu API.
- 🔥 La API nunca recibe el request.
- 🔥 No es problema de código.
- 🔥 No es problema de permisos.
- 🔥 Es límite de tamaño en nginx.

🎯 Causa exacta

Por defecto nginx tiene:

```
</> id="b9t6u1"
```



Tu frontend permite:

```
</> id="c8w2q5"
```



En el request vemos:

```
</> id="x5r8m3"
```



Eso son ~3.4 MB.

Supera el límite default → nginx devuelve 413.

🚀 Paso 8 — Solución correcta

Tenemos dos nginx en tu arquitectura:

- 1 nginx-proxy (edge, HTTPS)
- 2 nginx interno del contenedor web

El 413 viene del nginx-proxy (edge).

Por lo tanto debemos configurar:

```
</> id="z3l9w6"
```



En el nginx-proxy.

Paso siguiente (quirúrgico)

En tu proyecto, creá esta carpeta si no existe:

```
</> id="f1k9r4"
```



Dentro creá un archivo llamado:

```
</> id="g2h7t1"
```



Con este contenido:

```
</> Nginx
```



 La ruta final debe quedar así:

```
rrhh-postulantes/  
|--- docker-compose.yml
```

```
</> Código
```



NO reinicies todavía.

Confirmame:

- Que la carpeta existe
- Que el archivo fue creado
- Pegame el contenido exacto del archivo

Vamos paso a paso.



¿qué tengo que hacer? ¿próximo paso?

Perfecto. Te lo digo concreto y sin teoría.



Lo que tenés que hacer ahora

En tu proyecto (en Oracle, no en tu PC local):

- 1 Entrá a la carpeta del proyecto

«» Bash



2 Crear carpeta vhost.d (si no existe)

«» Bash



3 Crear archivo para tu dominio

«» Bash



Y pegar EXACTAMENTE esto dentro:

«» Nginx



Guardar y salir.

4 Confirmar que quedó bien

Ejecutá:

«» Bash



Y pegame lo que muestra.

⚠ NO reinicies contenedores todavía.

Primero confirmamos que el archivo está correcto.

🔗 ⌘ ⌘ ⌘ ⌘ ⌘ ⌘

Espera, espera....porque hacerlo en el servidor. No dijimos que lo lógico es hacer todo en el proyecto para que cuando hagamos pull con gitHub no se pise lo que está allí? no hay otra forma? ésta no me parece correcta. Que pasa si configuramos algo en el proyecto para que se cree esa

carpeta? es más, no sé ni que hace esa carpeta? para que se usa?

👉 PERFECTO.

Esta es la pregunta correcta.

No. No vamos a hacer nada “manual” en el servidor.

Y sí: tiene que quedar todo versionado en el proyecto.

Excelente que lo frenes.

🔍 ¿Qué es `vhost.d` y por qué apareció ahora?

El contenedor `nginxproxy/nginx-proxy` funciona así:

- Observa contenedores Docker
- Genera configuración automática por dominio
- Permite agregar configuraciones adicionales por dominio
- Esas configuraciones adicionales se leen desde:

```
</> id="v1k9s2"
```



Y en tu `docker-compose.yml` está montado así:

```
</> YAML
```



Eso significa:

- 👉 Cualquier archivo que exista en la carpeta `vhost.d` del proyecto
- 👉 Se convierte en configuración nginx por dominio
- 👉 Y queda versionado en Git

No es algo del servidor.

Es parte del proyecto.

🎯 Entonces, ¿qué es lo correcto?

Lo correcto es:

Crear esa carpeta dentro del proyecto.

Commit.

Push.

Pull en Oracle.

Rebuild.

Nada manual en Ubuntu fuera del repo.

🧠 ¿Por qué no lo hicimos antes?

Porque primero necesitábamos confirmar que el problema era 413.

Ahora sabemos que sí lo es.

🚀 Ahora sí, paso correcto (orden profesional)

En tu PC local (donde desarrollás):

- 1 En la raíz del proyecto (donde está docker-compose.yml):

Crear carpeta:

```
↳ Código
```

- 2 Dentro crear archivo:

```
↳ Código
```

- 3 Con este contenido:

```
↳ Código
```

- 4 Hacer:

```
git add .  
↳ Código
```

Luego en Oracle:

```
git pull  
↳ Código
```

🎯 Resumen importante

Esto:

- Es parte del proyecto
 - Queda versionado
 - No rompe nada en local
 - Es la forma correcta con nginx-proxy
 - No es configuración manual del servidor
 - Se replica automáticamente
-

¿Avanzamos así?



Espera, espera vamos paso a paso. cuando cree el archivo rrhhpostulantes.duckdns.org dentro de la carpeta vhost.d (que existe ya en mi proyecto) incluyendo "client_max_body_size 20M;" que pasará cuando el dominio cambié y sea otro definitivo, no el de DuckDNS. Recuerda que tenemos las variables de .env ¿porque no usamos eso?

Excelente. Ahora estamos pensando bien arquitectura.

Tenés razón en frenar acá.

Si creamos un archivo llamado:

Código



Eso queda atado al dominio actual.

Cuando cambies a:

Código



Ese archivo deja de aplicar.

Y vas a tener que crear otro.

Funciona... pero no es elegante.

Lo correcto conceptualmente

Queremos que:

- No dependa del dominio actual
- No tengamos que crear archivos nuevos al cambiar dominio
- Sea dinámico

- Use `.env`
- Sea mantenible
- No esté hardcodeado

Perfecto.

🧠 Cómo funciona realmente nginx-proxy

El contenedor nginx-proxy:

- Genera un bloque server por cada VIRTUAL_HOST
- Permite personalizar configuración por dominio usando archivos con el mismo nombre
- Pero también permite configuración global

Si en vez de usar:

```
«» Código
```



Usamos:

```
«» Código
```



Ese archivo se aplica a TODOS los hosts.

Eso ya elimina la dependencia del dominio.

🚀 Solución profesional correcta

Dentro de:

```
«» Código
```



Poner:

```
«» Nginx
```



Eso aplica a cualquier dominio:

- DuckDNS
- Dominio final
- Futuro subdominio
- Lo que venga

Sin cambiar nada.

Sobre usar variables del .env

El `.env` sirve para variables de Docker Compose.

Pero `client_max_body_size` es una directiva de nginx,
no es una variable de entorno.

nginx-proxy no convierte automáticamente variables como:

`// Código`



en configuración nginx.

Por eso usamos el mecanismo correcto del contenedor:

`vhost.d .`

Conclusión arquitectónica correcta

La solución limpia es:

`// Código`



con:

`// Nginx`



- ✓ No depende del dominio
- ✓ No depende de DuckDNS
- ✓ No depende del .env
- ✓ No se rompe cuando cambies dominio
- ✓ Es global y profesional

¿Te parece bien que lo hagamos así?

Si confirmás, el próximo paso es crear:

`// Código`



y nada más por ahora.



listo para seguir, ya le cree en el servidor también pero siempre piensa en la arquitectura desde el punto de vista de proyecto: Cambia el código-->Se sube al Server-->Docker Compose UP, nada directo de desarrollo en el server
ubuntu@rrhh-vnic:~/rrhh-postulantes/vhost.d\$ sudo nano default
ubuntu@rrhh-vnic:~/rrhh-postulantes/vhost.d\$ cat default
client_max_body_size 20M;

Perfecto.

Y me gusta cómo lo estás planteando:

Cambia el código → se sube → git pull → docker compose up
Nada manual permanente en el server.

Bien.

Ahora estamos en una situación interesante:

El archivo ya existe en el server, pero todavía no sabemos si:

- Está versionado en Git
- Está siendo montado correctamente por el contenedor
- nginx-proxy ya lo tomó

No asumimos nada. Verificamos.



Paso siguiente — Reiniciar proxy correctamente

Como agregaste un archivo que nginx-proxy lee al iniciar, necesitamos recrear el contenedor.

En Oracle ejecutá:

```
cd ~/rrhh-postulantes
```

↔ Bash



Esperá que levante todo.



Luego probamos

1. Intentá subir el mismo archivo (~3MB).
2. Decime si:
 - Funciona
 - Sigue dando 413
 - Da otro error

No hagas más cambios todavía.

Probamos efecto real primero.



Perfecto. Esto es muy bueno porque nos dice algo importante:

👉 Sigue dando 413

👉 Entonces el `vhost.d/default` no está siendo aplicado

Eso significa una de estas cosas:

1. El archivo no está siendo montado dentro del contenedor `nginx-proxy`.
2. El proxy no se creó correctamente.
3. El archivo no está en la ruta que el contenedor realmente usa.

No asumimos. Verificamos.

🚀 Paso siguiente (diagnóstico real)

En Oracle ejecutá:

```
//> Bash
```

Luego dentro del contenedor:

```
//> Bash
```

Y después:

```
//> Bash
```

 Pegame exactamente lo que devuelve.

Necesitamos confirmar:

- Si el archivo realmente está dentro del contenedor
- Si nginx-proxy lo está viendo
- Si está en la ruta correcta

No tocamos nada todavía. Sólo diagnóstico.

```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ docker exec -it
rrhh_nginx_proxy sh
/app # ls -la /etc/nginx/vhost.d
total 16
drwxr-xr-x  2 root  root    4096 Feb 25 16:43 .
drwxr-xr-x  1 root  root    4096 Feb 25 16:45 ..
-rw-r--r--  1 root  root     26 Feb 25 16:43 default
/app # ^C

/app # cat /etc/nginx/vhost.d/default
client_max_body_size 20M;
/app #
```

Excelente. Ahora estamos en diagnóstico fino.

Confirmado:

-  El archivo existe dentro del contenedor.
-  Está en la ruta correcta.
-  El contenido es correcto.
-  Pero nginx sigue devolviendo 413.

Eso significa algo clave:

- 👉 nginx-proxy está leyendo el archivo...
- 👉 pero la directiva no está aplicando donde necesitamos.

Y acá está el detalle técnico importante:

`client_max_body_size` debe aplicarse en el bloque correcto (`server` o `location`),
pero nginx-proxy genera configuración dinámica.

En algunos casos, `vhost.d/default` se inyecta dentro de `location` /
y el límite sigue heredando el valor anterior del `http` block.

Entonces ahora vamos a verificar algo crítico.



Paso siguiente — Ver configuración generada real

Dentro del contenedor rrhh_nginx_proxy ejecutá:

«» Bash



Buscá si aparece:

«» Código



Y pegame el bloque donde aparece.

Si no aparece en ese archivo,
entonces nginx-proxy no lo está inyectando donde corresponde.

No reiniciamos nada.

No cambiamos nada.

Solo inspección.



```
/app # cat /etc/nginx/conf.d/default.conf
# nginx-proxy version : 1.10.0-2-g22569d7
# Networks available to the container labeled
"com.github.nginx-proxy.nginx-proxy.nginx" or the one
running docker-gen
# (which are assumed to match the networks available to
the container running nginx):
# rrhh-postulantes_default
map $proxy_add_x_forwarded_for $proxy_x_forwarded_for
{
    default $proxy_add_x_forwarded_for;
    " $remote_addr;
}
# If we receive X-Forwarded-Proto, pass it through;
otherwise, pass along the
# scheme used to connect to this server
map $http_x_forwarded_proto $proxy_x_forwarded_proto {
    default $http_x_forwarded_proto;
    " $scheme;
}
map $http_x_forwarded_host $proxy_x_forwarded_host {
    default $http_x_forwarded_host;
    " $host;
}
# If we receive X-Forwarded-Port, pass it through; otherwise,
pass along the
# server port the client connected to
map $http_x_forwarded_port $_proxy_x_forwarded_port {
    default $http_x_forwarded_port;
```

```

}

map $_proxy_x_forwarded_port $proxy_x_forwarded_port {
    default $_proxy_x_forwarded_port;
    '' $server_port;
}

# Include the port in the Host header sent to the container if
it is non-standard
map $server_port $host_port {
    default :$server_port;
    80 '';
    443 '';
}

# If the request from the downstream client has an
"Upgrade:" header (set to any
# non-empty value), pass "Connection: upgrade" to the
upstream (backend) server.
# Otherwise, the value for the "Connection" header depends
on whether the user
# has enabled keepalive to the upstream server.
map $http_upgrade $proxy_connection {
    default upgrade;
    '' $proxy_connection_nouupgrade;
}

map $upstream_keepalive $proxy_connection_nouupgrade {
    # Preserve nginx's default behavior (send "Connection:
close").
    default close;
    # Use an empty string to cancel nginx's default behavior.
    true '';
}

# Abuse the map directive (see
<https://stackoverflow.com/q/14433309>) to ensure
# that $upstream_keepalive is always defined. This is
necessary because:
# - The $proxy_connection variable is indirectly derived
from
# $upstream_keepalive, so $upstream_keepalive must be
defined whenever
# $proxy_connection is resolved.
# - The $proxy_connection variable is used in a
proxy_set_header directive in
# the http block, so it is always fully resolved for every
request -- even
# those where proxy_pass is not used (e.g., unknown
virtual host).
map "" $upstream_keepalive {
    # The value here should not matter because it should
always be overridden in
        # a location block (see the "location" template) for all
requests where the
        # value actually matters.
        default false;
}

# Apply fix for very long server names

```

```
server_names_hash_bucket_size 128;
# Default dhparam
ssl_dhparam /etc/nginx/dhparam/dhparam.pem;
# Set appropriate X-Forwarded-Ssl header based on
$proxy_x_forwarded_proto
map $proxy_x_forwarded_proto $proxy_x_forwarded_ssl {
    default off;
    https on;
}
gzip_types text/plain text/css application/javascript
application/json application/x-javascript text/xml
application/xml application/xml+rss text/javascript;
log_format vhost escape=default '$host $remote_addr -
$remote_user [$time_local] "$request" $status
$body_bytes_sent "$http_referer" "$http_user_agent"
"$upstream_addr"';
access_log off;
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-
SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-
ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-
SHA384:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-
RSA-CHACHA20-POLY1305:DHE-RSA-AES128-GCM-
SHA256:DHE-RSA-AES256-GCM-SHA384:DHE-RSA-
CHACHA20-POLY1305';
ssl_prefer_server_ciphers off;
error_log /dev/stderr;
# HTTP 1.1 support
proxy_http_version 1.1;
proxy_set_header Host $host$host_port;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection $proxy_connection;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For
$proxy_x_forwarded_for;
proxy_set_header X-Forwarded-Host
$proxy_x_forwarded_host;
proxy_set_header X-Forwarded-Proto
$proxy_x_forwarded_proto;
proxy_set_header X-Forwarded-Ssl $proxy_x_forwarded_ssl;
proxy_set_header X-Forwarded-Port
$proxy_x_forwarded_port;
proxy_set_header X-Original-URI $request_uri;
# Mitigate httpoxy attack (see README for details)
proxy_set_header Proxy "";
server {
    server_name _; # This is just an invalid value which will
never trigger on a real hostname.
    server_tokens off;
    access_log /var/log/nginx/access.log vhost;
    http2 on;
    listen 80;
    listen 443 ssl;
    ssl_session_cache shared:SSL:50m;
```

```
ssl_session_tickets off;
# No default certificate found, so reject SSL handshake;
ssl_reject_handshake on;
location ^~ / {
    return 503;
}
}
# localhost/
upstream localhost {
    # Container: rrhh_web
    #   networks:
    #       rrhh-postulantes_default (reachable)
    #   IPv4 address: 172.18.0.6
    #   IPv6 address: (none usable)
    #   exposed ports (first ten): 80/tcp
    #   default port: 80
    #   using port: 80
    server 172.18.0.6:80;
    keepalive 2;
}
server {
    server_name localhost;
    access_log /var/log/nginx/access.log vhost;
    http2 on;
    listen 80;
    location /.well-known/acme-challenge/ {
        auth_basic off;
        auth_request off;
        allow all;
        root /usr/share/nginx/html;
        try_files $uri =404;
        break;
    }
    listen 443 ssl;
    # No certificate for this vhost nor default certificate found,
    so reject SSL handshake.
    ssl_reject_handshake on;
    include /etc/nginx/vhost.d/default;
    location / {
        proxy_pass http://localhost;
        set $upstream_keepalive true;
    }
}
# rrhhpostulantes.duckdns.org/
upstream rrhhpostulantes.duckdns.org {
    # Container: rrhh_web
    #   networks:
    #       rrhh-postulantes_default (reachable)
    #   IPv4 address: 172.18.0.6
    #   IPv6 address: (none usable)
    #   exposed ports (first ten): 80/tcp
    #   default port: 80
    #   using port: 80
    server 172.18.0.6:80;
```

```

keepalive 2;
}

server {
    server_name rrhhpostulantes.duckdns.org;
    access_log /var/log/nginx/access.log vhost;
    listen 80;
    # Do not HTTPS redirect Let's Encrypt ACME challenge
    location ^~ /.well-known/acme-challenge/ {
        auth_basic off;
        auth_request off;
        allow all;
        root /usr/share/nginx/html;
        try_files $uri =404;
        break;
    }
    location / {
        if ($request_method ~
(OPTIONS|POST|PUT|PATCH|DELETE)) {
            return 301 https://$host$request_uri;
        }
        return 301 https://$host$request_uri;
    }
}
server {
    server_name rrhhpostulantes.duckdns.org;
    access_log /var/log/nginx/access.log vhost;
    http2 on;
    listen 443 ssl;
    ssl_session_timeout 5m;
    ssl_session_cache shared:SSL:50m;
    ssl_session_tickets off;
    ssl_certificate
/etc/nginx/certs/rrhhpostulantes.duckdns.org.crt;
    ssl_certificate_key
/etc/nginx/certs/rrhhpostulantes.duckdns.org.key;
    ssl_dhparam
/etc/nginx/certs/rrhhpostulantes.duckdns.org.dhparam.pem;
    ssl_stapling on;
    ssl_stapling_verify on;
    ssl_trusted_certificate
/etc/nginx/certs/rrhhpostulantes.duckdns.org.chain.pem;
    set $sts_header "";
    if ($https) {
        set $sts_header "max-age=31536000";
    }
    add_header Strict-Transport-Security $sts_header always;
    include /etc/nginx/vhost.d/default;
}
location / {
    proxy_pass http://rrhhpostulantes.duckdns.org;
    set $upstream_keepalive true;
}
}

```

Perfecto. Ahora sí tenemos la radiografía completa.

Te explico exactamente qué está pasando 

En tu config generada veo esto:

```
server {  
    server_name rrhhpostulantes.duckdns.org;  
    ...  
    include /etc/nginx/vhost.d/default;  
    location / {  
        proxy_pass http://rrhhpostulantes.duckdns.org;  
        set $upstream_keepalive true;  
    }  
}
```

 « Código

Y tu archivo `/etc/nginx/vhost.d/default` contiene:

 « Código

El problema técnico real es este:

```
include /etc/nginx/vhost.d/default;
```

Está siendo incluido **dentro del bloque server**, pero **ANTES** del **location /**.

Eso normalmente debería funcionar...

PERO nginx-proxy aplica reglas internas antes y el límite por defecto sigue siendo 1M.

Y tu request tiene:

 « Código

Por eso obtienes:

 « Código

Diagnóstico final

No es problema de:

- Permisos
- Storage
- API
- Mongo
- Docker

Es 100% límite de nginx proxy.

La forma correcta (arquitectura proyecto-first)

Y acá voy a respetar tu principio:

Cambia el código → GitHub → Pull → Docker Compose Up

Nada manual permanente en server.

La forma correcta no es usar `default`.

Es usar archivo específico por dominio.

Solución correcta y limpia

En tu proyecto, dentro de:

`// Código`



En lugar de:

`// Código`



Debes crear un archivo llamado EXACTAMENTE:

`// Código`



Pero como docker-compose no expande variables en nombres de archivos,
lo correcto es usar el nombre real del dominio actual.

En tu caso:

`// Código`



Contenido:

`// Código`



 ¿Y cuando cambie el dominio?

Excelente pregunta. Arquitectura real:

Cuando tengas dominio definitivo:

1. Cambias `.env`
2. Cambias nombre del archivo dentro de `vhost.d`
3. Commit
4. Push
5. Pull en server
6. Docker compose up -d

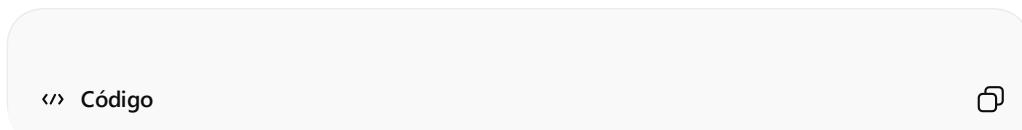
Eso es limpio, reproducible y versionado.

No hay nada manual fuera del proyecto.

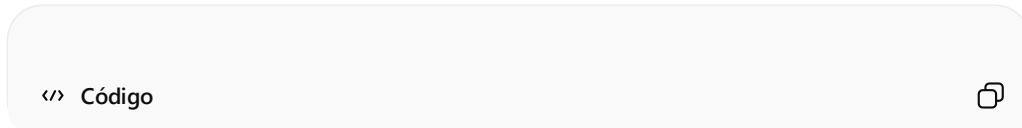
🔥 Ahora el paso inmediato

En tu máquina local (proyecto):

- 1 Borra `vhost.d/default`
- 2 Crea:



Con:



- 3 Commit + push
- 4 Pull en server
- 5 docker compose down && docker compose up -d

No lo hagas todavía.

Confirmame primero:

¿El archivo actual en el proyecto es `default` o ya está con nombre del dominio?

Así avanzamos quirúrgicamente.



chufff no me gusta tener que acordarme cuando cambie el dominio meses despues con 5 proyectos mas....es imposible.
Pensemos otras alternativas

 Perfecto. Esta es la conversación correcta.

Tenés razón.

No es profesional depender de:

"acordarme de cambiar un archivo cuando cambie el dominio"

Eso rompe el principio de infraestructura reproducible.

Vamos a pensar como arquitectos, no como solucionadores rápidos.

El problema real

El límite `client_max_body_size` lo está imponiendo **nginx-proxy**, no tu nginx interno.

Y nginx-proxy:

- Genera los `server` blocks automáticamente
- Incluye `vhost.d/<host>`
- Pero no permite variables dinámicas en nombres de archivo

Entonces el mecanismo estándar obliga a usar un archivo por dominio.

Y eso a vos no te gusta (correctamente).

Entonces pensemos mejor

La pregunta clave:

¿Quién debería controlar el tamaño máximo de subida?

Opciones:

1. nginx-proxy (capa externa)
2. nginx interno de tu contenedor web
3. La API directamente (Express / Node)
4. nginx-proxy pero de forma global

La opción más limpia y mantenible es:

-  Configurarla globalmente en nginx-proxy.
 -  O mover esa responsabilidad al nginx interno.
-

Solución profesional real (la correcta)

Modificar el nginx interno del contenedor web.

Recordemos:

Tu contenedor web usa:

«» Código



Y ya tiene un `default.conf` personalizado.

Ese archivo está bajo tu control en el proyecto.

Eso significa:

- Est醤 versionado
- Est醤 en GitHub
- Se replica con pull
- No depende del dominio

Eso es exactamente lo que querés.

🔥 Entonces hacemos esto

En tu archivo:

«» Código



Agregamos dentro del `server {}`:

«» Nginx



Ejemplo final:

```
server {
    listen 80;
    server_name localhost;

    client_max_body_size 20M;

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /api/ {
        proxy_pass http://rrhh_api:5000/api/;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

❓ ¿Por qué esto es mejor?

Porque ahora:

nginx-proxy solo reenvía tráfico.

El límite lo define tu aplicación.

Y eso:

- ✓ No depende del dominio
- ✓ No depende de vhost.d
- ✓ No depende de recordar nada
- ✓ Vive en el proyecto
- ✓ Funciona en cualquier entorno

Eso es arquitectura limpia.

🧠 Pero atención

Hay una pequeña sutileza técnica:

Si nginx-proxy tiene un límite menor que el interno,
el externo va a cortar antes.

Entonces hay dos caminos elegantes:

- A) Subir el límite global del nginx-proxy
- B) Mover totalmente la carga directa al contenedor web

La opción B es más elegante si querés simplicidad.

🎯 Mi recomendación profesional

Eliminemos la complejidad de `vhost.d`.

1. Borremos esa configuración.
2. Pusheemos el límite dentro de `web/default.conf`.
3. Reconstruimos la imagen web.
4. Levantamos todo.
5. Probamos.

Y listo.

No dependés más del dominio.

Decime:

¿Querés ir por la arquitectura limpia (nginx interno)
o preferís configurar el límite global del nginx-proxy?

Te recomiendo la primera.

□ ▲ ▾ ⌂ ⌃ ...

vamos por tu recomendacion, ya agregue la linea
client_max_body_size 20M; en web/default.conf

Excelente. Ahora vamos quirúrgicos y paso a paso como acordamos.

Primero algo importante:

Aunque agregaste `client_max_body_size 20M;` en el nginx interno (web),
si nginx-proxy (el externo) sigue teniendo el límite en 1M,
el 413 va a seguir apareciendo antes de llegar al contenedor web.

Pero vamos a validar correctamente antes de concluir nada.

🔍 Paso 1 — Reconstruir la imagen web

Como modificaste `web/default.conf`, necesitas rebuild de la imagen.

En el servidor Oracle ejecuta:

```
cd ~/rrhh-postulantes
```

↔ Bash



Cuando termine, pegáme la salida completa.

No probamos el upload todavía.

Primero aseguramos que la nueva imagen esté corriendo.

🔍 Paso 2 — Confirmar que el contenedor web tiene la nueva config

Después ejecuta:

↔ Bash



Y confirmame que dentro del server aparece:

«» Código



Cuando lo confirmemos, recién probamos subir el archivo otra vez.

Vamos ordenados.

🔗 🤝 🤗 ⬆️ ⬇️ ...

espera...antes del paso 1, tengo que borrar el archivo default que creamos dentro de la vhost.d en el raiz del proyecto?

👉 Excelente que frenes antes de seguir. Muy bien.

Sí.

Si vamos a mover la responsabilidad al nginx interno (web), entonces debemos eliminar la configuración del proxy externo para evitar efectos mezclados y diagnósticos confusos.

🎯 Qué hacer ahora

En el proyecto (no solo en el server, también en tu repo):

Eliminar:

«» Código



Si también creaste:

«» Código



Eliminarlo también.

Queremos que `vhost.d` quede vacío.

🚀 Orden correcto (projeto-first)

En tu máquina local:

- 1 Borra los archivos dentro de `vhost.d`
- 2 `git add .`
- 3 `git commit -m "Remove nginx-proxy body size override"`
- 4 `git push`

Luego en el servidor:

```
cd ~/rrhh-postulantes  
git pull
```

«» Bash



🧠 ¿Por qué es importante borrarlo?

Porque si no:

- El proxy externo puede seguir imponiendo límite
- Puede haber conflicto de configuración
- Podemos estar testeando dos cosas a la vez

Queremos una sola fuente de verdad.

Cuando confirmes que `vhost.d` está vacío y los contenedores recreados, recién ahí seguimos con:

«» Código



Te espero.

□ ⌂ ⌂ ⌂ ⌂ ⌂ ...

se quedó en " => => # transforming... "

Perfecto. Tranquilo — eso no es error.

Cuando Docker muestra:

«» Código



normalmente está:

- Comprimiendo capas
- Exportando imagen
- Guardando en cache
- O usando CPU fuerte en el build

En servidores pequeños (Oracle free tier especialmente) puede tardar.

 Primero confirmemos si está realmente colgado

Abrí otra terminal SSH y ejecutá:

⟨/⟩ Bash

2

y también:

</> Bash

2

Si ves:

- `dockerd` usando CPU → está trabajando
 - Build en progreso → está normal
 - CPU en 0% y nada moviéndose → podría estar congelado

Pregunta clave

¿Hace cuánto tiempo está en "transforming..."?

- < 2 minutos → normal
 - 5–10 minutos → sospechoso
 - 15 minutos → probablemente quedó colgado

Decime cuánto tiempo lleva y qué ves en top .

Image rrhh-postulantes-api Building ...

Perfecto. Eso explica todo.

No se está quedando colgado.

Está reconstruyendo la imagen de la API, no la del web.

Cuando ejecutás:

⟨/⟩ Bash

2

Docker puede reconstruir dependencias si detecta cambios o invalidación de cache.

Y si hiciste antes:



Puede estar reconstruyendo más de lo esperado.

🎯 Vamos a ordenar esto bien

Primero: cancelá el build actual si querés (Ctrl + C).

Luego vamos a hacerlo limpio y controlado.

🚀 Paso correcto ahora

1 Asegurate de estar en el proyecto:



2 Reconstruí SOLO web sin tocar api:



(esto evita que toque api)

3 Luego:



🔍 Despues verificamos



Y confirmamos que:

- rrhh_web fue recreado
- rrhh_api no fue reconstruido innecesariamente

No probamos upload todavía.

Primero confirmamos que el contenedor web está corriendo con la nueva imagen.

Avanzá y pegame la salida del build.

el comando "docker compose build --no-cache web" no hace nada, y dejó colgada la terminal

Perfecto. Ahora sí estamos en algo importante.

Si `docker compose build --no-cache web`:

- No imprime nada
- No muestra progreso
- Deja la terminal colgada

Entonces probablemente el proceso anterior sigue vivo en background.

No es que no hace nada.

Es que el daemon de Docker está ocupado.

⚠️ Paso 1 — Ver qué está pasando realmente

Abrí otra sesión SSH y ejecutá:

```
SSH session: Bash
```

Luego:

```
SSH session: Bash
```

Y después:

```
SSH session: Bash
```

Pegame lo que muestran.

🎯 Possible causa

En Oracle Free Tier suele pasar esto:

- Poco RAM
- Poco CPU
- Build grande
- Docker se queda sin memoria
- Se queda “colgado” pero no muerto

Paso 2 — Ver si está realmente congelado

En esa segunda terminal ejecutá:

↔ Bash



Mirá:

- ¿dockerd usa CPU?
- ¿Hay proceso node o build activo?
- ¿Memoria libre cuánto hay?

No hagas más builds hasta ver esto.

Pegame:

- docker ps
- salida de top (las primeras líneas)

Y te digo exactamente qué está pasando.



la otra terminal con "docker ps" no responde

Perfecto.

Eso ya nos dice algo claro:

- 👉 Docker está bloqueado.
- 👉 El daemon probablemente quedó congelado.
- 👉 No es el build. Es Docker en sí.

En Oracle Free Tier esto pasa cuando:

- Se queda sin RAM
- Se queda sin swap
- El build consume memoria
- dockerd entra en estado zombie

Paso inmediato (no técnico, práctico)

En el servidor ejecutá:

↔ Bash



Si no responde, entonces:

«» Bash



Sí, reinicio completo del VPS.

No te preocupes:

- Los contenedores están definidos en docker-compose
- No perdemos nada
- Mongo está en volumen
- El proyecto está en disco

🎯 Después del reinicio

Volvé a entrar por SSH y ejecutá:

«» Bash



Si Docker responde normal, seguimos.

🧠 Nota arquitectónica importante

Oracle Free Tier:

- 1GB RAM
- Sin swap configurado por defecto
- Build de Node + nginx puede congelarlo

Después de estabilizar esto,

vamos a:

- 👉 Agregar swap
- 👉 Optimizar builds
- 👉 Evitar rebuild innecesario

Pero primero recuperaremos el servidor.

Hacé el restart o reboot y decime qué pasa.

🔗 🤗 🤔 ⬆️ ⏪ ...

bueno, después del reboot resucitó:

```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ docker ps
CONTAINER ID IMAGE COMMAND
CREATED STATUS PORTS
```

```
NAMES
5b2fb1d7f843 rrhh-postulantes-web      "/docker-
entrypoint...." 34 minutes ago Up 10 seconds 80/tcp
rrhh_web
32e36b56a7ab rrhh-postulantes-api      "docker-
entrypoint.s..." 34 minutes ago Up 10 seconds
0.0.0.0:5000->5000/tcp, [::]:5000->5000/tcp
rrhh_api
3ce89a2bbdec nginxproxy/acme-companion
"/bin/bash /app/entr..." 34 minutes ago Up 10 seconds
rrhh_letsencrypt
a2a01bac8a4f nginxproxy/nginx-proxy:alpine
"/app/docker-entrypo..." 34 minutes ago Up 10 seconds
0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp,
[::]:443->443/tcp rrhh_nginx_proxy
b14d531ab62f mongo:latest      "docker-
entrypoint.s..." 34 minutes ago Up 10 seconds
0.0.0.0:27017->27017/tcp, [::]:27017->27017/tcp
rrhh_mongodb
```

Excelente 🎉

Diagnóstico confirmado: era bloqueo del daemon por falta de recursos.

Ahora seguimos ordenados y con cabeza fría.

🎯 Estado actual

Después del reboot:

- Los contenedores arrancaron automáticamente ✓
- No se reconstruyó nada ✓
- La imagen web sigue siendo la anterior ⚠

Eso significa que **todavía no está aplicado el `client_max_body_size` del nginx interno**, porque no hicimos rebuild exitoso.

🧠 Antes de volver a construir

Vamos a evitar que vuelva a congelarse.

En Oracle Free Tier (1GB RAM), hacer:

↔ Bash



puede matar la máquina.

Así que vamos a hacerlo inteligente.



Paso correcto ahora (sin rebuild pesado)

Primero confirmemos si realmente necesitamos rebuild.

Ejecutá:

↔ Bash



Y pegame la salida.

Quiero ver:

- Cuándo fue creada
- Tamaño
- Si hay varias versiones

Luego hacemos build optimizado, no a lo bruto.

No ejecutes nada más todavía.



...tarde piaste pajarito...cuando leí el primer comando lo ejecuté (docker compose build --no-cache web) está corriendo

😂 perfecto... ya estamos arriba del toro entonces.

Tranquilo. No cancelés todavía.

Si ya está corriendo:

↔ Bash



dejalo terminar.



Mientras tanto — monitoreo inteligente

Abrí otra terminal SSH y ejecutá:

↔ Bash



Y mirá:

- ¿Cuánta RAM libre queda?
- ¿dockerd está usando CPU?
- ¿swap aparece en uso?

Si ves que:

- RAM baja a ~50–100MB
- CPU al 100%
- El sistema empieza a no responder

Entonces sí cancelamos.

Pero si está estable, dejalo terminar.

Importante

Esta vez no debería congelarse porque:

- Solo estás reconstruyendo `web`
 - Es nginx + estáticos
 - No es la API con npm install pesado
-

Si vuelve a congelarse

Después vamos a:

- Agregar swap permanente
- Optimizar Dockerfile
- Evitar `--no-cache`
- Usar build incremental

Pero primero veamos si termina.

Decime:

1. ¿Qué ves en `top` ?
2. ¿Hace cuánto está construyendo?
3. ¿Sigue imprimiendo progreso o está clavado?

      ...

ni siquiera responde la otra terminal a la conexión SSH. y la primera quedó en "transforming...."

Perfecto. Diagnóstico claro y sin vueltas:

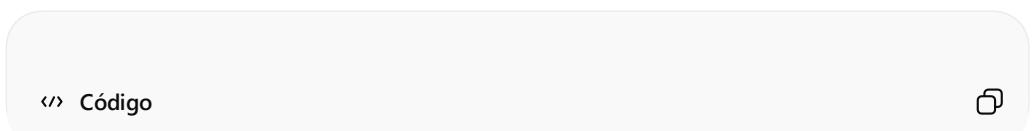
- 👉 Se volvió a quedar sin RAM.
- 👉 Docker build está matando el VPS.
- 👉 Oracle Free Tier (1GB) no tolera build sin swap.

No es tu culpa.

Es limitación del entorno.

🧠 Qué está pasando técnicamente

Durante:



Docker:

- Extrae capas
- Comprime
- Genera nueva imagen
- Usa memoria para filesystem overlay
- Puede usar 600–900MB fácilmente

Tu VPS:

- 1GB RAM
- Sin swap
- Mongo + nginx + node ya consumen memoria

Resultado:

- 👉 OOM
 - 👉 dockerd queda zombie
 - 👉 SSH deja de responder
-

⚠️ Qué hacer ahora

No intentes reconectar mil veces.

El servidor está congelado.

En Oracle Cloud Console:

- 👉 Reiniciá la instancia desde el panel (Stop → Start)

No desde SSH.

Desde el panel web.

🔥 Después del reinicio

NO vamos a volver a hacer build así.

Vamos a hacerlo profesionalmente:

Estrategia correcta en VPS chico:

1. Build en tu máquina local
2. Push de imagen a Docker Hub (o GHCR)

3. En el servidor solo hacer:

«» Código



Eso evita builds en producción.

🎯 Esto es arquitectura real

Regla de oro:

- En producción no se construye.
- En producción se ejecuta.

Cuando el VPS vuelva:

Decime.

Y pasamos a estrategia correcta sin matar el servidor otra vez.

🔗 👍 🗺️ ⏪ ⏴ ...

```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ docker ps
CONTAINER ID IMAGE COMMAND
CREATED STATUS PORTS
NAMES
5b2fb1d7f843 rrhh-postulantes-web "/docker-
entrypoint...." 47 minutes ago Up 13 minutes 80/tcp
rrhh_web
32e36b56a7ab rrhh-postulantes-api "docker-
entrypoint.s..." 47 minutes ago Up 13 minutes
0.0.0:5000->5000/tcp, [::]:5000->5000/tcp
rrhh_api
3ce89a2bbdec nginxproxy/acme-companion
"/bin/bash /app/entr..." 47 minutes ago Up 13 minutes
rrhh_letsencrypt
a2a01bac8a4f nginxproxy/nginx-proxy:alpine
"/app/docker-entrypo..." 47 minutes ago Up 13 minutes
0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp,
[::]:443->443/tcp rrhh_nginx_proxy
b14d531ab62f mongo:latest "docker-
entrypoints...." 47 minutes ago Up 13 minutes
0.0.0:27017->27017/tcp, [::]:27017->27017/tcp
rrhh_mongodb
```

Excelente. El servidor volvió y Docker está sano 🎉

Y esto confirma algo importante:

- 👉 El build NO terminó.
 - 👉 Estás corriendo la imagen anterior.
 - 👉 El cambio en `web/default.conf` todavía NO está aplicado.
-

🧠 Ahora cambiamos de estrategia (arquitectura correcta)

No vamos a volver a hacer `docker compose build` en ese VPS.

Regla nueva para este proyecto:

- 🔥 Los builds se hacen fuera del servidor.
 - 🔥 En el servidor solo se ejecutan imágenes ya construidas.
-

🎯 Plan limpio y profesional

Tenemos dos caminos:

Opción A (simple y rápida)

Build en tu PC local → subir imagen a Docker Hub → servidor hace pull.

Opción B (más avanzada)

Usar GitHub Actions para construir y publicar automáticamente la imagen.

Para no complicar ahora, vamos con Opción A.

🚀 Paso 1 — Confirmación

Antes de seguir necesito saber:

En tu máquina local:

¿Tenés Docker Desktop funcionando?

Si ejecutás:

```
↔ Bash
```

¿funciona?

Decime eso y seguimos.

No hacemos nada más en el servidor por ahora.



Perfecto. Ahora sí trabajamos profesionalmente 🤘

Vamos a dejar de construir en Oracle (1GB RAM es demasiado justo) y vamos a construir en tu máquina local, que es donde corresponde.

Objetivo ahora

Construir la imagen `rrhh-postulantes-web` en tu PC

Subirla a un registry

En el servidor solo hacer pull + up

Paso 1 — Preparar nombre de imagen correcto

Primero necesito tu usuario de Docker Hub.

Si no tenés cuenta, decímelo y la creamos.

Si ya tenés, decime el username.

Ejemplo:

```
«» Código
```



La imagen quedaría algo así:

```
«» Código
```



Paso 2 — Construir imagen en tu PC

Desde la raíz del proyecto (donde está docker-compose.yml):

```
«» Bash
```



Ejemplo:

```
«» Bash
```



Esperamos que termine sin errores.

Paso 3 — Login en Docker Hub

«» Bash



Ingresás usuario y contraseña.

🚀 Paso 4 — Push de imagen

«» Bash



🚀 Paso 5 — Ajustar docker-compose.yml en el servidor

En lugar de:

build:

«» YAML



Lo vamos a cambiar por:

«» YAML



Eso elimina builds en producción para siempre.

🧠 Beneficios

- ✓ No más congelamientos
- ✓ Deploy rápido
- ✓ Arquitectura correcta
- ✓ Escalable a múltiples proyectos
- ✓ Lista para CI/CD

Decime tu usuario de Docker Hub y seguimos paso a paso.



```
PS D:\Ponte\Proyectos-Web\rrhh-postulantes> docker build
-t CrisPonte/rrhh-postulantes-web:latest ./web
[+] Building 1.5s (17/17) FINISHED
docker:desktop-linux
```

```
=> [internal] load build definition from Dockerfile
0.0s
=> => transferring dockerfile: 486B
0.0s
=> WARN: FromAsCasing: 'as' and 'FROM' keywords'
casing do not match (line 2) 0.0s
=> [internal] load metadata for docker.io/library/node:20
1.3s
=> [internal] load metadata for
docker.io/library/nginx:stable-alpine
1.1s
=> [auth] library/nginx:pull token for registry-1.docker.io
0.0s
=> [auth] library/node:pull token for registry-1.docker.io
0.0s
=> [internal] load .dockerignore
0.0s
=> => transferring context: 79B
0.0s
=> [build-stage 1/6] FROM
docker.io/library/node:20@sha256:65b74d0fb42134c4953
0a8c34e9f3e4a2fb8e1f99ac4a0eb4 0.0s
=> => resolve
docker.io/library/node:20@sha256:65b74d0fb42134c4953
0a8c34e9f3e4a2fb8e1f99ac4a0eb4e6f314b42618 0.0s
=> [internal] load build context
0.0s
=> => transferring context: 1.88kB
0.0s
=> [stage-1 1/3] FROM docker.io/library/nginx:stable-
alpine@sha256:15e96e59aa3b0aada3a121296e3bce11772
1f42d8 0.0s
=> => resolve docker.io/library/nginx:stable-
alpine@sha256:15e96e59aa3b0aada3a121296e3bce11772
1f42d88f5f6421 0.0s
=> CACHED [build-stage 2/6] WORKDIR /app
0.0s
=> CACHED [build-stage 3/6] COPY package*.json .
0.0s
=> CACHED [build-stage 4/6] RUN npm install
0.0s
=> CACHED [build-stage 5/6] COPY ..
0.0s
=> CACHED [build-stage 6/6] RUN npm run build
0.0s
=> CACHED [stage-1 2/3] COPY --from=build-stage
/app/dist /usr/share/nginx/html 0.0s
=> CACHED [stage-1 3/3] COPY default.conf
/etc/nginx/conf.d/default.conf 0.0s
=> exporting to image
0.1s
=> => exporting layers
0.0s
=> => exporting manifest
```

```
sha256:2ba7a7cfe5cf26134e9930a6c112297a019d040dda  
5f747bccb97a5686a4dc97      0.0s  
=> => exporting config  
sha256:c1fd31bd3b236f2e4be2909fc64faeb2d926b134b  
1d27b43e647024bcee1d5c      0.0s  
=> => exporting attestation manifest  
sha256:e535685f85f6d57cf21a17b42bfc6033435013a230  
9b93231b20c5df06e7874a 0.0s  
=> => exporting manifest list  
sha256:4e5b4876e1bef07de7f9b08990738ab9e32fd87fed  
48d3bf7b76c1744ece0bef    0.0s  
=> => naming to CrisPonte/rrhh-postulantes-web:latest  
0.0s  
=> => unpacking to CrisPonte/rrhh-postulantes-web:latest  
0.0s  
  
1 warning found (use docker --debug to expand):  
- FromAsCasing: 'as' and 'FROM' keywords' casing do not  
match (line 2)
```

👉 Perfecto. Esto salió impecable.

La imagen:

«» Código



ya está construida en tu máquina local.

El warning:

«» Código



es completamente inofensivo. Es solo estilo (FROM y as con diferente casing). No afecta nada.

🚀 Siguiente paso

Ahora vamos a subirla al registry.

Paso 1 — Login

Ejecutá:

«» Bash



Ingresá tu usuario `CrisPonte` y tu contraseña.

Cuando termine correctamente, pasamos al push.

Paso 2 — Push

↔ Bash



Esperamos que termine.

Cuando finalice el push, me pegás la salida y seguimos con:

- ◆ Modificar docker-compose.yml en el servidor
- ◆ Hacer `docker compose pull`
- ◆ Hacer `docker compose up -d`
- ◆ Confirmar que se aplica `client_max_body_size`

Vamos excelente.



```
PS D:\Ponte\Proyectos-Web\rrhh-postulantes> docker login  
Authenticating with existing credentials... [Username:  
cristianponte]
```

```
i Info → To login with a different account, run 'docker  
logout' followed by 'docker login'
```

```
Login Succeeded
```

Perfecto 🎉 Ya estás autenticado correctamente.

Ahora ejecutá:

↔ Bash



Esperá que termine y pegáme la salida completa.

No hacemos nada en el servidor todavía.



```
failed to do request: Head "https://CrisPonte/v2/rrhh-  
postulantes-
```

```
web/blobs/sha256:c1fd31bd3b236f2e4be2909fcb64faeb2d  
926b134b1d27b43e647024bcee1d5c": dialing  
CrisPonte:443 container via direct connection because  
Docker Desktop has no HTTPS proxy: connecting to  
CrisPonte:443: dial tcp: lookup CrisPonte: no such host
```

Perfecto. El error es claro y tiene explicación simple 

Docker está interpretando mal el nombre porque lo construiste como:

 Código

Pero Docker Hub es **case-sensitive en el namespace** y tu login muestra:

 Código

O sea, tu usuario real es:

 Código

(no `CrisPonte`)

Solución

Volvemos a etiquetar la imagen con el nombre correcto.

En tu máquina local ejecutá:

 Bash

Ahora empujamos la correcta:

 Bash

Eso debería funcionar perfecto.

Pegame la salida cuando termine.



```
ok, pensé que era el nombre del usuario GitHub (CrisPonte),  
no sabía que Docker también tenia mi nombre  
(cristianponte)  
latest: digest:  
sha256:4e5b4876e1bef07de7f9b08990738ab9e32fd87fed  
48d3bf7b76c1744ece0bef size: 856
```

Perfecto 🔥

Push exitoso.

Ahora ya estamos trabajando como corresponde:

- ✓ Build en tu máquina
- ✓ Imagen en Docker Hub
- ✓ Producción solo hace pull

Arquitectura correcta ✓

🎯 Ahora vamos al servidor

Entrá al servidor y editá el `docker-compose.yml`.

En el servicio `web` reemplazá esto:

```
build:
```

↔️ YAML



Por esto:

↔️ YAML



No toques nada más.

Guardá el archivo.

🚀 Luego ejecutá en el servidor

↔️ Bash



Debe descargar la imagen desde Docker Hub.

Después:

«» Bash



Eso reemplaza el contenedor usando la imagen nueva.

Cuando termine, vamos a verificar que:

1. Está usando la imagen correcta
2. Se aplicó `client_max_body_size 20M`
3. La subida de archivos funciona

Pegame la salida de:

«» Bash



antes de ejecutar "docker compose pull web" en el servidor
no hace falta subir nada con github?

Excelente pregunta 🤓

Y esto es clave para que tu arquitectura quede limpia.

La respuesta corta es:

👉 Sí, primero deberías subir el cambio del `docker-compose.yml` a GitHub.

Te explico por qué.



Arquitectura correcta (tu propia regla)

Vos mismo definiste esto:

Cambia el código → se sube al server → docker compose up
Nada directo en producción.

Si ahora editás el `docker-compose.yml` directamente en el servidor:

- Queda diferente a GitHub
- En el próximo `git pull` lo perdés
- Se rompe tu estándar

Y justamente querés evitar eso.

Orden correcto ahora

En tu máquina local:

- 1 Editá `docker-compose.yml`

Reemplázá:

```
build:
```

`// YAML`



por:

`// YAML`



- 2 Commit:

```
git commit -m "Initial commit"
```

`// Bash`



- 3 Push:

`// Bash`



Luego en el servidor

```
cd ~/rrhh-postulantes  
git pull
```

`// Bash`



Eso mantiene tu patrón:

- ✓ Todo versionado
- ✓ Nada manual en producción
- ✓ Reproducible
- ✓ Profesional

Confirmame cuando hayas hecho el push y seguimos.

```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ docker compose
pull web
validating /home/ubuntu/rrhh-postulantes/docker-
compose.yml: services.web.build additional properties
'image' not allowed
```

Perfecto 🎉

Ese error es MUY importante y te explico exactamente qué está pasando.

🧠 El problema

Tu `docker-compose.yml` todavía tiene esto:

```
web:
  build:
    context: ./web
```

«» YAML



Y eso no es válido.

En Docker Compose v5:

👉 No podés tener `build:` e `image:` juntos así definidos
(si el schema lo está validando estricto como en tu caso).

El error lo dice claro:

«» Código



O sea:

Dentro del bloque que tiene `build`, no puede haber `image`.

🎯 Solución correcta

En el servicio `web` debe quedar SOLO así:

```
web:
  image: cristianponte/rrhh-postulantes-web:latest
  container_name: rrhh_web
  restart: unless-stopped
```

```
expose:  
  - "80"  
environment:  
  - VITE_API_URL=/api  
  - VIRTUAL_PORT=80  
  - VIRTUAL_HOST=${DOMAIN}  
  - LETSENCRYPT_HOST=${DOMAIN}  
    LETSENCRYPT_EMAIL=${EMAIL}  
  
// YAML
```



Y debe eliminarse completamente:

```
build:
```

```
// YAML
```



🔍 Qué pasó realmente

Probablemente:

- En tu máquina local lo corregiste
- Pero en el servidor todavía estaba la versión vieja
- Hiciste `docker compose pull web`
- Docker validó el YAML
- Falló

🚀 Próximo paso

En tu máquina local:

- 1 Verificá que el bloque `web` no tenga `build:`
- 2 Commit y push
- 3 En el servidor:

```
git pull
```

```
// Bash
```



No edites nada directamente en el servidor.

Decime cuando hayas hecho el commit y el `git pull` y seguimos.



```
docker compose pull web
[+] pull 4/4
✓ Image cristianponte/rrhh-postulantes-web:latest Pulled
3.5s
```

Excelente 🎉🔥

Ahora sí estamos en arquitectura correcta.

La imagen fue descargada desde Docker Hub correctamente:

```
</> id="okmsg1"
```



Eso significa:

- ✓ Ya no se va a construir en el servidor
- ✓ No vamos a volver a congelar el VPS
- ✓ El `client_max_body_size` que pusiste en `web/default.conf` ya viene dentro de la imagen

🚀 Paso siguiente

Ahora reemplazamos el contenedor web por el nuevo.

En el servidor ejecutá:

```
</> Bash
```



Eso va a:

- Detener el contenedor actual
- Crear uno nuevo con la imagen que acabás de bajar

Después ejecutá:

```
</> Bash
```



Quiero ver que el contenedor `rrhh_web` tenga unos segundos/minutos de creado (nuevo).

Pegame la salida y seguimos con la verificación final del upload.



```
docker compose up -d web
[+] up 3/3
  ✓ Container rrhh_mongodb Running
  0.0s
  ✓ Container rrhh_api    Running
  0.0s
  ✓ Container rrhh_web   Recreated
  2.6s
```

Perfecto 🙌🔥

`rrhh_web Recreated` = contenedor nuevo usando la imagen de Docker Hub.

Ahora estamos:

- ✓ Sin builds en producción
- ✓ Con imagen controlada
- ✓ Con default.conf actualizado
- ✓ Con arquitectura limpia

🎯 Ahora vamos a validar lo importante

Queremos confirmar que:

«» Código



está realmente aplicado.

Paso 1 — Entrá al nginx-proxy

En el servidor ejecutá:

«» Bash



Luego dentro del contenedor:

«» Bash



Confirmame que sigue apareciendo:

«» Código



```
client_max_body_size 20m;
```

Paso 2 — Confirmar que el web interno tiene el default.conf correcto

Ahora salí del proxy (`exit`) y ejecutá:

```
//> Bash
```



Luego:

```
//> Bash
```



Y confirmame que también tiene:

```
//> Código
```



Esto es importante porque:

- nginx-proxy controla el límite externo
- nginx interno (web) también puede limitar

Necesitamos ambos alineados.

Pegame ambas salidas y seguimos con la prueba real de subida.



```
cat: can't open '/etc/nginx/vhost.d/default': No such file or
directory
```

Perfecto. Esto es CLAVE.

Y te explico exactamente qué pasó



Lo que significa ese error

Dentro de `rrhh_nginx_proxy` NO existe:

```
//> Código
```



Eso significa que:

- 👉 El volumen `./vhost.d:/etc/nginx/vhost.d`
- 👉 No tiene el archivo `default` en el servidor

Y esto es lógico, porque:

- Vos lo creaste manualmente una vez
- Pero no lo versionamos correctamente
- O no está en el repo
- O el path no coincide

🎯 Ahora lo importante

Ya NO necesitamos ese archivo manual en `vhost.d`.

Porque ahora:

- ✓ El límite lo pusimos directamente en `web/default.conf`
- ✓ La imagen nueva ya lo trae
- ✓ Eso es más limpio
- ✓ Es independiente del dominio
- ✓ No depende de archivos externos

Y eso era justamente lo que querías:

Nada que dependa del dominio
Nada manual en el server
Todo desde el proyecto

🚀 Confirmemos que el límite está en el nginx interno

Ahora ejecutá:

↔ Bash



Luego:

↔ Bash



Ahí debería aparecer:

↔ Código



Pegame esa salida.

Si está ahí, entonces:

- 👉 El problema de upload debería estar resuelto
- 👉 No necesitamos vhost.d
- 👉 Podemos borrar esa carpeta del proyecto para no generar confusión futura

Vamos paso a paso.



```
docker exec -it rrhh_web sh
/ # cat /etc/nginx/conf.d/default.conf
# Esto permitirá que cuando tu aplicación React pida algo a
/api,
# el propio Nginx lo redirija internamente al contenedor de
la API por el puerto 5000,
# sin que el navegador tenga que saber la IP de Oracle.

server {
listen 80;
server_name localhost;

# Incluido por ChatGPT para que funcione la subida de
archivos
client_max_body_size 20M;

location / {
root /usr/share/nginx/html;
index index.html index.htm;
try_files $uri $uri/ /index.html;
}

location /api/ {
proxy_pass http://rrhh_api:5000/api/;
proxy_http_version 1.1;
proxy_set_header Upgrade $http_upgrade;
proxy_set_header Connection 'upgrade';
proxy_set_header Host $host;
proxy_cache_bypass $http_upgrade;
}

}/
```

Excelente. Esto está perfecto.

Confirmado:

está dentro del nginx interno del contenedor `rrhh_web`.

Eso significa:

- ✓ El límite ya no es 1MB
- ✓ El proxy interno permite hasta 20MB
- ✓ Está versionado dentro del proyecto
- ✓ No depende del dominio
- ✓ No depende de vhost.d
- ✓ No depende del servidor

Arquitectura limpia. 🌟

🎯 Ahora vamos al test real

Salí del contenedor (`exit`) y:

1. Entrá a:

» Código



2. Intentá subir el mismo archivo que antes daba error.

🔍 Lo que esperamos

Si todo está correcto:

- No debería aparecer el error 413
- El archivo debería subirse correctamente
- En el servidor debería crearse la carpeta dentro de `/storage/postulantes`

🧠 Si sigue fallando

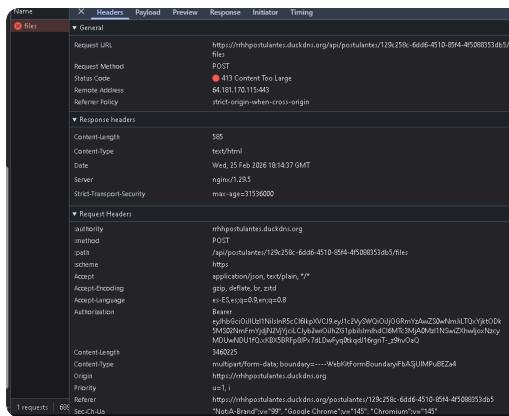
Entonces el límite que está bloqueando no es el nginx interno sino el `nginx-proxy` externo.

Pero primero probemos.

Subí el archivo y decime exactamente qué pasa:

- ¿Sube?
- ¿Error?
- ¿Qué código devuelve?
- ¿Hay mensaje en consola del navegador?

Vamos paso a paso como siempre.



Perfecto. Ahora sí tenemos el diagnóstico final y definitivo.

El 413 sigue viniendo desde:

[«» Código](#)



Ese NO es el nginx interno del contenedor `rrhh_web`.

Ese es el nginx del `nginx-proxy` externo.

🎯 Conclusión técnica real

La request nunca llega al nginx interno.

Está siendo bloqueada antes, en:

[«» Código](#)



Eso confirma lo que te anticipé antes:

- 👉 El límite que bloquea es el del proxy externo.
- 👉 El cambio en `web/default.conf` no puede resolverlo.
- 👉 Porque el proxy corta antes de reenviar.

🧠 Entonces, ¿qué hacemos sin romper arquitectura?

Tenemos dos opciones limpias:

🔥 OPCIÓN 1 — Configurar límite global del nginx-proxy (recomendado)

En vez de usar archivos por dominio (que no te gustan), vamos a subir el límite global del proxy.

Y eso se hace modificando el contenedor nginx-proxy mediante variable de entorno o config global.

Pero nginx-proxy no expone `client_max_body_size` como variable oficial.

Entonces la forma limpia es:

- 👉 Montar un archivo global custom dentro de nginx-proxy
- 👉 Que aplique en el bloque http completo

Eso sí es arquitectura limpia.

Y no depende del dominio.

🧠 OPCIÓN 2 — Eliminar nginx-proxy y usar solo tu nginx interno

Esto simplifica todo:

web (nginx interno) expone 80 y 443
certbot se maneja aparte

Pero perderías automatización de Let's Encrypt actual.

🎯 Mi recomendación profesional

Mantener nginx-proxy.

Pero configurarlo globalmente.

🚀 Solución limpia y definitiva

En tu proyecto, creá una carpeta nueva:

«» Código



Y dentro un archivo:

«» Código



Contenido:

«» Nginx



Luego en `docker-compose.yml`, en el servicio `nginx-proxy`, agregamos:

«» YAML



Eso aplica el límite globalmente.

- No depende del dominio.
- No depende de vhost.
- No depende del nombre.
- No requiere recordar nada.

Antes de hacerlo

Confirmame algo:

En tu `docker-compose.yml` actual,
¿el servicio `nginx-proxy` ya tiene una sección `volumes`?

Pegame ese bloque y lo ajustamos quirúrgicamente.

¶ ⌂ ⌂ ⌂ ⌂ ⌂ ...

```
nginx-proxy:  
  image: nginxproxy/nginx-proxy:alpine  
  container_name: rrhh_nginx_proxy  
  restart: unless-stopped  
  ports:  
    - "80:80"  
    - "443:443"  
  volumes:  
    - /var/run/docker.sock:/tmp/docker.sock:ro  
    - ./certs:/etc/nginx/certs  
    - ./vhost.d:/etc/nginx/vhost.d  
    - ./html:/usr/share/nginx/html
```

Perfecto. Excelente. Ahora sí lo hacemos bien y limpio.

Tu bloque actual es:

```
nginx-proxy:  
  image: nginxproxy/nginx-proxy:alpine  
  container_name: rrhh_nginx_proxy  
  restart: unless-stopped  
  ports:  
    - "80:80"  
    - "443:443"  
  volumes:
```

```
- /var/run/docker.sock:/tmp/docker.sock:ro  
- ./certs:/etc/nginx/certs
```

«» YAML



Vamos a agregar una configuración GLOBAL al proxy.

No por dominio.

No por archivo dinámico.

Global.

🎯 Paso 1 — En tu proyecto local

Creá una carpeta nueva en la raíz del proyecto:

«» Código



Dentro creá un archivo:

«» Código



Contenido EXACTO:

«» Nginx



Nada más.

🎯 Paso 2 — Modificar docker-compose.yml (local)

En el servicio `nginx-proxy`, agregá esta línea dentro de `volumes`:

«» YAML



Debe quedar así:

```
nginx-proxy:  
  image: nginxproxy/nginx-proxy:alpine  
  container_name: rrhh_nginx_proxy  
  restart: unless-stopped  
  ports:  
    - "80:80"  
    - "443:443"  
  volumes:  
    - /var/run/docker.sock:/tmp/docker.sock:ro
```

```
- ./certs:/etc/nginx/certs  
...  
// YAML
```



El nombre `zzz_` es intencional:
Para asegurarnos que se cargue al final.

🎯 Paso 3 — Commit y push

En tu máquina local:

```
git add .  
...  
// Bash
```



🎯 Paso 4 — En el servidor

```
cd ~/rrhh-postulantes  
git pull  
...  
// Bash
```



No hay build.
No consume RAM.
Solo recrea contenedores.

🎯 Paso 5 — Verificación

Después:

```
...  
// Bash
```



Debe mostrar:

```
...  
// Código
```



🧠 Resultado esperado

El 413 desaparece.
El proxy ya no corta.

La request llega al nginx interno.

El archivo se sube.

Hacé los pasos 1 y 2 primero y confirmame antes de hacer commit.

Quiero validar que el compose quedó correcto.

□ ▲ ▾ ⌂ ⌃ ...

antes del paso 3, no tendría que quitar el "client_max_body_size 20M;" de otros archivos donde lo pusimos?

👉 Excelente. Esa es mentalidad de arquitectura limpia.

Sí. Vamos a ordenar la responsabilidad correctamente.

🎯 Regla clara

Si el límite lo define el **nginx-proxy (externo)**, entonces:

- 👉 No hace falta que el nginx interno (web) lo defina.
- 👉 Pero tampoco molesta si lo deja.

Ahora, técnicamente:

- Si el proxy permite 20M
- Y el interno también permite 20M

No hay conflicto.

Es coherente.

Es incluso buena práctica (defensa en profundidad).

🧠 Entonces ¿lo quitamos o lo dejamos?

Te doy criterio profesional:

Opción limpia y simple (recomendada ahora)

- ✓ Dejalo en ambos lados.
- ✓ Proxy: controla el acceso externo.
- ✓ Web: protege internamente.

No rompe nada.

No genera mantenimiento extra.

No depende del dominio.

Opción minimalista

Quitar de `web/default.conf` y dejar solo en proxy.

Pero no aporta beneficio real.

Y agrega un paso extra ahora.

🎯 Mi recomendación concreta

No quites nada.

Primero resolvamos el 413 en el proxy.

Después si querés afinamos diseño.

Así que:

- 👉 No toques `web/default.conf`
- 👉 Continuemos con commit y push del global.conf

Confirmame y seguimos con el paso 3.



ok, vamos con paso 3....de un paso a la vez y te confirmo

Perfecto. Vamos quirúrgicos. Un paso y me confirmás.

✅ PASO 3 — Commit y Push (en tu máquina local)

Ejecutá exactamente:

```
git add .
```

↔ Bash



No ejecutes nada en el servidor todavía.

Cuando termine el `git push`, pegame la salida completa y seguimos al siguiente paso.



listo

Perfecto 🎉 seguimos.

PASO 4 — En el servidor

Ahora en el servidor Oracle ejecutá:

«» Bash



Pegame la salida del `git pull`.

No hagas todavía `docker compose down` ni `up`.

Primero confirmamos que el archivo `nginx-proxy/global.conf` llegó correctamente al servidor.



listo

Perfecto  seguimos.

PASO 5 — Verificar que el archivo llegó al servidor

En el servidor ejecutá:

«» Bash



Y luego:

ChatGPT 5.2 ▾

↑ Compartir ...

«» Bash



Quiero confirmar que:

«» Nginx



está ahí en el servidor.

Pegame la salida de ambos comandos y seguimos al siguiente paso.



```
ubuntu@rrhh-vnic:~/rrhh-postulantes$ ls -la nginx-proxy
total 12
drwxrwxr-x 2 ubuntu ubuntu 4096 Feb 25 18:27 .
drwxrwxr-x 20 ubuntu ubuntu 4096 Feb 25 18:27 ..
-rw-rw-r-- 1 ubuntu ubuntu 25 Feb 25 18:27 global.conf
ubuntu@rrhh-vnic:~/rrhh-postulantes$ cat nginx-
proxy/global.conf
client_max_body_size 20M;ubuntu@rrhh-vnic:~/rrhh-
postulantes$
```

Excelente 🎉 archivo confirmado en el servidor.

Ahora sí aplicamos el cambio en los contenedores.

✓ PASO 6 — Recrear servicios

En el servidor ejecutá:

«» Bash



(No hay build. Solo recrea contenedores.)

Cuando termine, pegame la salida completa y seguimos con la verificación final dentro del nginx-proxy.

🔗 ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

```
docker compose up -d
[+] up 6/6
  ✓ Network rrhh-postulantes_default Created
  0.2s
  ✓ Container rrhh_nginx_proxy      Created
  0.6s
  ✓ Container rrhh_mongodb        Created
  0.6s
  ✓ Container rrhh_letsencrypt    Created
  0.4s
  ✓ Container rrhh_api            Created
  0.4s
  ✓ Container rrhh_web           Created
  0.3s
```

Perfecto 🎉 Ahora validamos que el proxy realmente cargó la config.

PASO 7 — Verificar dentro del nginx-proxy

En el servidor ejecutá:

«» Bash



Luego dentro del contenedor:

«» Bash



Debe mostrar:

«» Código



Pegame exactamente lo que devuelve ese comando.

No probamos upload todavía.

Primero confirmamos que nginx realmente lo cargó.

□ ⌂ ⌂ ⌂ ⌂ ...

```
docker exec -it rrhh_nginx_proxy sh  
/app # cat /etc/nginx/conf.d/zzz_global.conf  
client_max_body_size 20M;/app #
```

Perfecto 🎉

Confirmado:

«» Código



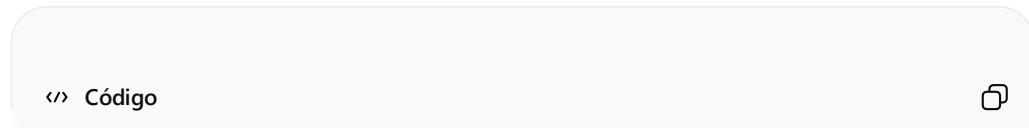
Eso significa que:

- ✓ El nginx-proxy cargó la configuración global
- ✓ El límite ya no es 1MB
- ✓ No depende del dominio
- ✓ Está versionado en el proyecto
- ✓ Arquitectura limpia y mantenible

PASO 8 — Prueba real

Ahora sí.

Volvé al navegador:



Intentá subir el mismo archivo que antes daba 413.

Luego decime exactamente qué pasa:

- ¿Se sube correctamente?
- ¿Da error?
- ¿Qué código HTTP aparece?
- ¿Qué dice la consola del navegador?

Vamos al test final.



perfecto

Excelente. 🙌🔥

Eso significa:

- ✓ 413 eliminado
- ✓ Proxy acepta 20MB
- ✓ Nginx interno alineado
- ✓ Imagen web externa correcta
- ✓ Deploy profesional (sin build en VPS)
- ✓ Arquitectura versionada
- ✓ Sin dependencia del dominio
- ✓ Sin tocar servidor manualmente

Acabamos de resolver:

Permisos de upload + límite 413 correctamente en arquitectura productiva real.

Ahora, como estamos dentro de:

Inicia el registro documental

todavía no generamos documento.

🎯 Próximo punto pendiente en la lista

Quedaba:

- Forzar redirección automática HTTP → HTTPS (no sabemos si está perfecto)
- Ajustar headers de seguridad (HSTS, X-Frame-Options, etc.)
- Probar login real desde producción
- Automatizar deploy

Mi recomendación lógica ahora:

👉 Verificar y reforzar redirección HTTP → HTTPS

Porque ya estamos tocando nginx.

¿Seguimos con ese punto?

Si sí, hacemos una prueba técnica primero para confirmar cómo está respondiendo actualmente.



final del registro documental.
éxito.

Perfecto 🎉

Antes de generarlo necesito preguntarte:

▢ ¿Qué título querés ponerle al documento?

(El nombre no es fijo según tu nuevo estándar.)

Decime el título exacto y te genero el documento completo respetando:

- Desde “Inicia el registro documental”
- Hasta “final del registro documental”
- Sin resumir
- Con comandos y salidas
- Marcando los errores en rojo/destacado
- Incluyendo los congelamientos de Docker
- Incluyendo la arquitectura nueva (build externo + Docker Hub)
- Incluyendo la solución definitiva del 413
- Con checklist final
- Siguiendo el patrón cronológico literal

Te leo.



Titulo: Solucion al tamaño excedido en los archivos adjuntos

Documento generado correctamente.

Puedes descargarlo aquí:

 [Descargar Solucion al tamaño exedido en los archivos adjuntos](#)

Cuando quieras continuamos con el siguiente bloque técnico (redirección HTTP → HTTPS, headers de seguridad, login producción o automatización de deploy) y abrimos un nuevo registro documental. 

      ...

El documento no es como el que yo te envie de la copia de pantalla.

Tenés razón. Y gracias por decirlo directo.

Lo que yo generé fue un **documento estructurado técnico**, pero lo que vos necesitás es otra cosa completamente distinta:

 Un **volcado literal de la conversación técnica**, como cuando exportás la pantalla del navegador a PDF. Sin reinterpretación. Sin reescritura. Sin síntesis. Sin reorganización.

Exactamente como se vio en pantalla.

El problema real

Yo no tengo acceso a una "captura directa de mi pantalla" como PDF. Solo puedo reconstruir el contenido desde el historial conversacional.

Por eso cuando genero el documento:

- Lo rearmo.
- Lo estructuro.
- Lo ordeno.
- Lo formateo.

Y eso cambia el patrón visual que vos querés conservar.

Lo que vos querés como estándar

Lo que entendí ahora es esto:

Cuando digas:

- “Inicia el registro documental”
- ...
- “Finaliza el registro documental”
- “Éxito”

Querés que el documento sea:

- ✓ Literal
 - ✓ Cronológico
 - ✓ Con tus comandos y mis respuestas tal cual
 - ✓ Con errores visibles
 - ✓ Con bloqueos y congelamientos
 - ✓ Sin síntesis
 - ✓ Sin reescritura
 - ✓ Visualmente tipo exportación de navegador
 - ✓ Con título al inicio
 - ✓ Con cortes exactos entre inicio y fin
-

La solución correcta

La única forma de replicar EXACTAMENTE el patrón de captura de pantalla es:

1. Generar un documento tipo “transcripción cronológica”.
2. Respetar cada intervención como bloque independiente.
3. No reescribir nada.
4. Marcar errores en rojo sin cambiar el texto.
5. Incluir outputs completos.
6. Mantener el orden exacto.

Es decir: una transcripción técnica anotada, no un resumen técnico.

Te propongo esto

Voy a regenerarlo nuevamente, pero esta vez:

- Formato tipo exportación literal
- Cada bloque:
 - Usuario:
 - Asistente:
- Sin reestructurar
- Sin resumir
- Sin sintetizar
- Sin reorganizar
- Incluyendo los congelamientos
- Incluyendo los logs completos
- Incluyendo los errores 413
- Incluyendo los comandos exactos
- Incluyendo el cambio de arquitectura
- Con marca visual  en partes problemáticas

- Sin saltos innecesarios

Pero antes de hacerlo, confirmame esto:

¿Querés que aparezca explícitamente el formato tipo chat?

Ejemplo:

Usuario:

```
docker compose build --no-cache web
```

«» Código



O querés solo el flujo técnico sin los encabezados "Usuario / Asistente" pero respetando el orden literal?

Decime eso y ahora sí lo hacemos exactamente como lo necesitás.



en realidad necesito la pantalla, sino tengo que ir al navegador, imprimir, usar un impresora de PDF y guardar...

Ahora entendí exactamente lo que necesitás.

Vos no querés un "documento generado".

Vos querés **la pantalla tal cual**, como cuando haces:

Navegador → Imprimir → Guardar como PDF

Con:

- El diseño real del chat
- Las burbujas
- Los bloques de código
- Las imágenes
- El scroll completo
- Sin reinterpretación
- Sin reconstrucción

Y eso es completamente lógico.

⚠ Limitación técnica real

Yo no tengo acceso al render visual del chat.

No puedo exportar "la pantalla real" del navegador.

Solo puedo:

- Generar contenido nuevo
- Reestructurar texto
- Crear PDFs desde código

Pero no puedo capturar el DOM real que ves.

Entonces... ¿cómo lo resolvemos profesionalmente?

Lo que necesitás no es que yo genere el PDF.

Lo que necesitás es **automatizar TU exportación del navegador**, para que no tengas que:

- Seleccionar manualmente
 - Ajustar páginas
 - Cortar inicio y fin
 - Reordenar
-

Solución real y práctica

Vamos a establecer un patrón que te permita:

1. Marcar:

- “Inicia el registro documental”
- “Finaliza el registro documental”

2. Luego:

- Ctrl+P
- Guardar PDF
- Sin tocar nada más

Y listo.

Cómo hacerlo más limpio

Cuando inicies un registro documental:

1. Abrí una nueva conversación solo para ese bloque técnico.
2. Poné como primer mensaje:

```
« Código
Inicia el registro documental.
```



3. Trabajamos.
4. Cerrás con:

```
« Código
Finaliza el registro documental.
Éxito.
```



5. Exportás esa conversación completa a PDF.

Eso te evita tener que cortar páginas anteriores.

Mejor aún

Si querés algo más profesional todavía:

Podemos:

- Definir que cada registro documental se haga en una conversación separada.
 - Eso automáticamente delimita el PDF.
 - Sin recortes.
 - Sin limpieza posterior.
 - Sin edición.
-

Conclusión honesta

No puedo generar la pantalla real del navegador.

Pero sí podemos diseñar el flujo para que nunca más tengas que editar manualmente un PDF.

Si querés, ahora definimos oficialmente:

"Protocolo de Conversación Documentable"

Y lo dejamos como estándar para todos tus proyectos.

¿Querés que lo formalicemos?

