GitHub Repo DockerHub API DockerHub Frontend

# To-Do App – Conteinerización y Orquestación (Compose · Swarm · Kubernetes KIND) - Práctica Nº1 – Arquitectura de Microservicios con Retry, Circuit Breaker y Load Balancer

**Alumno:** Elmer Alan Cornejo Quito
**Usuario Docker Hub:** alancor
**Repositorio:** AlanCornejoQ/todo-app

## Estructura del repositorio

```
./todo-app/
├── api/
├── compose.yml
├── docker-compose.yml
├── frontend
│   ├── default.conf
│   ├── Dockerfile
│   └── index.html
├── k8s/
├── kind-config.yaml
├── LICENSE
├── load-balancer
│   ├── Dockerfile
│   └── nginx.conf
├── README.md
├── README.pdf
├── services
│   ├── tasks-service
│   │   ├── app.py
│   │   ├── Dockerfile
│   │   └── requirements.txt
│   └── users-service
│       ├── app.py
│       ├── Dockerfile
│       └── requirements.txt
└── stack-deploy.yml
```

## Ejecución del proyecto

1. Clonar o descomprimir el repositorio.
2. Construir y levantar todos los servicios:

```
docker compose up -d --build
```

3. Verificar estado:

```
docker compose ps
```

# Pruebas de funcionamiento

## 1. Frontend y CRUD

Abrir http://localhost:5173

- Crear una nueva tarea.
- Marcarla como completada.
- Usar **Actualizar** para refrescar la lista.

## 2. Retry y Circuit Breaker

1. **Apagar una réplica** del servicio de tareas:

   ```
   docker compose stop tasks-a-1
   ```

   El sistema sigue funcionando gracias al Load Balancer.
2. **Apagar ambas réplicas:**

   ```
   docker compose stop tasks-a-2
   ```

   El endpoint /me/tasks responde con fallback:

   ```
   {"fallback":true,"tasks":[],"message":"tasks no disponible"}
   ```

3. **Volver a encenderlas:**

   ```
   docker compose start tasks-a-1 tasks-a-2
   ```

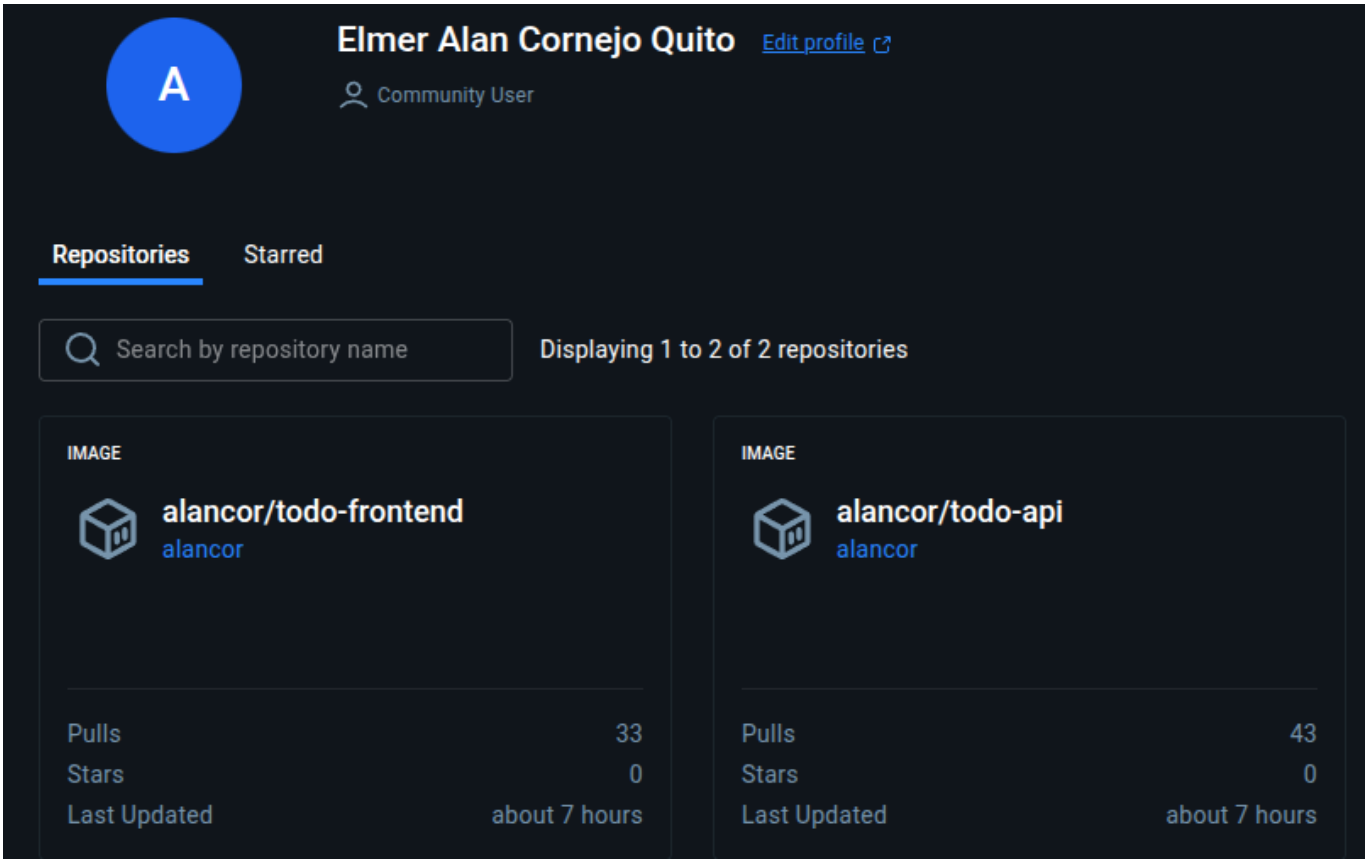
   se restablece automáticamente.

# 1 Descripción

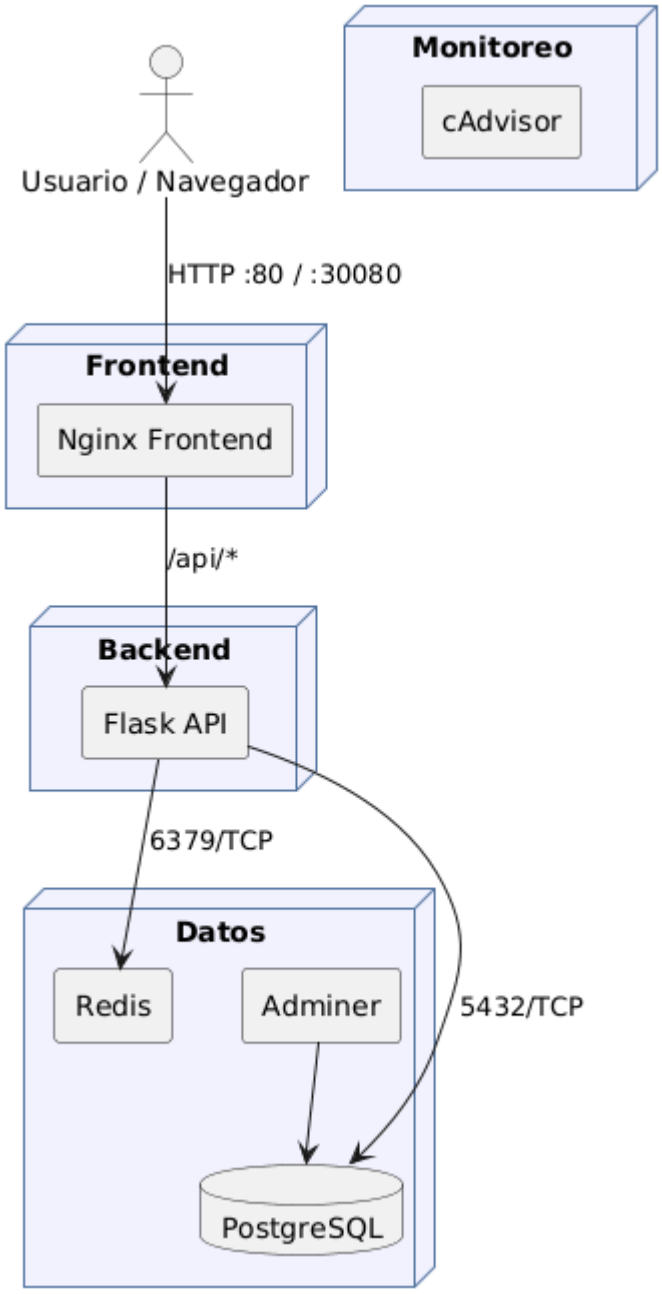
Aplicación **To-Do** modular con seis servicios:

| Servicio | Rol | Imagen |
|---|---|---|
| `frontend` | UI Nginx (HTML + JS) | `alancor/todo-frontend:v1.1.0` |
| `api` | Backend Flask + Gunicorn | `alancor/todo-api:v1.1.0` |
| `db` | Base de datos PostgreSQL 16-alpine | `postgres:16-alpine` |
| `redis` | Cache/mensajería | `redis:7-alpine` |
| `adminer` | GUI de BD | `adminer:latest` |
| `cadvisor` | Monitoreo de contenedores (Compose) | `gcr.io/cadvisor/cadvisor:v0.49.1` |

Repositorio público de imágenes en Docker Hub (namespace **alancor**):




## 2 Arquitectura




- Red interna entre servicios (app-net)
- Variables de entorno para credenciales y conexiones
- Volumen persistente para la base de datos

---

## 3 Repositorio público de imágenes

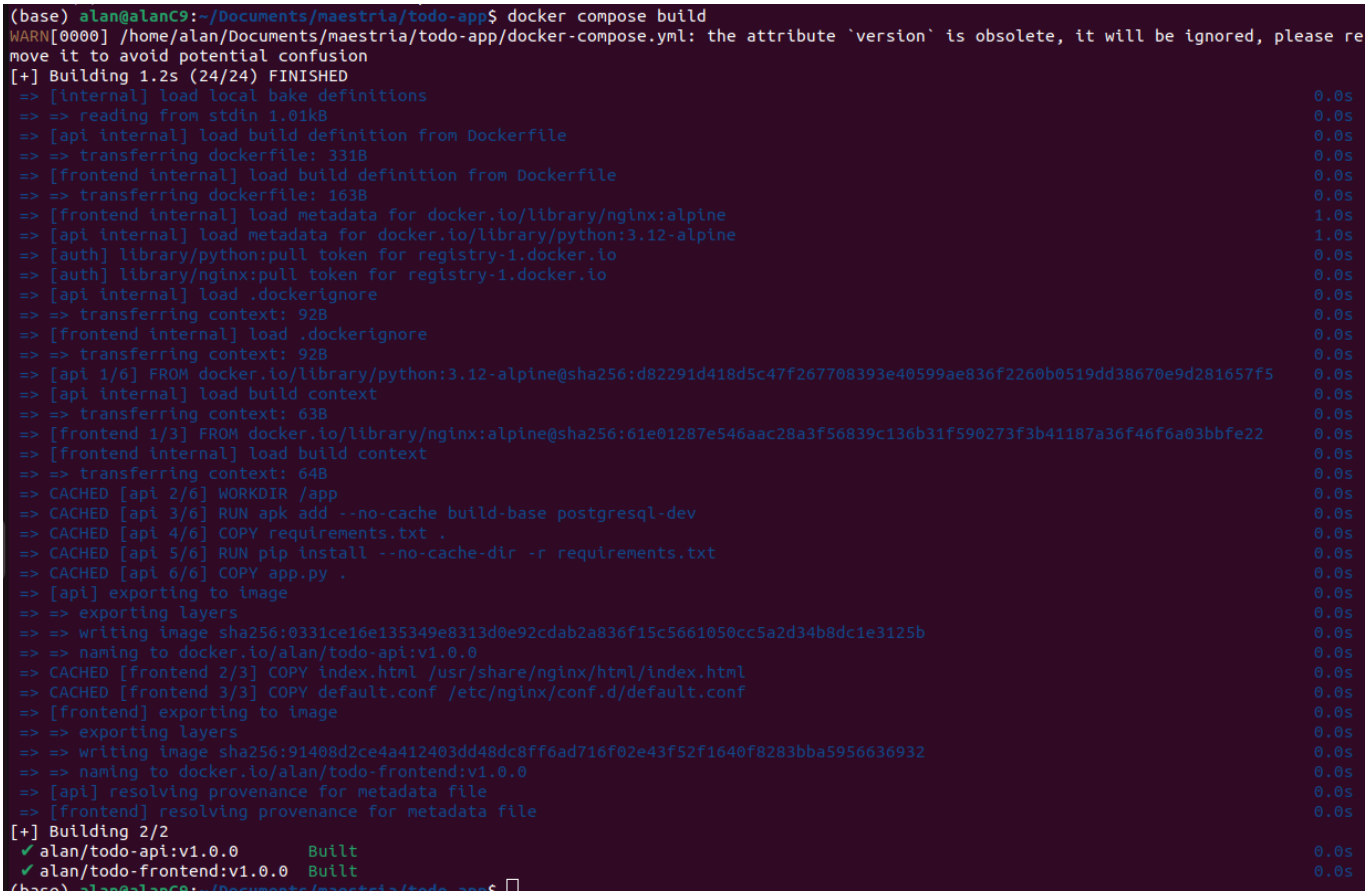
| Imagen | Versión | Descripción |
|---|---|---|
| `alancor/todo-api:v1.0.0` | Versión inicial | CRUD de tareas básicas |
| `alancor/todo-api:v1.1.0` | Actualización | Endpoint `/` con información de versión |
| `alancor/todo-frontend:v1.0.0` | Versión inicial | Interfaz web To-Do |
| `alancor/todo-frontend:v1.1.0` | Actualización | Encabezado “To-Do v1.1.0” |

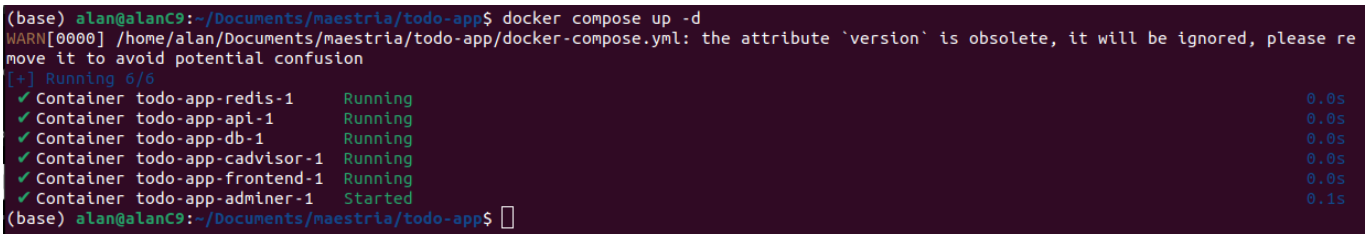
> Imágenes personalizadas construidas con Dockerfiles propios (base **Alpine**), optimizadas con `.dockerignore` y publicadas en el repositorio público **Docker Hub**.
> Cumple los requerimientos de **conteinerización**, **versionamiento** y **repositorio público**.

---

## 4 Despliegue local – *Docker Compose*

```
docker compose build
docker compose up -d
docker compose ps
```





**Accesos**

- Frontend → http://localhost:8080
- Adminer → http://localhost:8081
- cAdvisor → http://localhost:8082

> Se levanta toda la solución en red personalizada `app-net`, con volumen persistente `dbdata` y variables desde `.env`.

---

## 5 Despliegue en *Docker Swarm*

```
docker swarm init
printf "todosecret" | docker secret create db_password -
printf "production" | docker config create app_env -
docker stack deploy -c stack-deploy.yml todoapp
docker stack services todoapp
docker service ps todoapp_api
```

**Salida relevante**

```
(base) alan@alanC9:~/Documents/maestria/todo-app$ docker stack services todoapp
ID             NAME               MODE         REPLICAS   IMAGE                          PORTS
znjrri24bhtu   todoapp_adminer    replicated   1/1        adminer:latest                 *:8081->8080/tcp
3ro4xkqlsn7w   todoapp_api        replicated   2/2        alancor/todo-api:v1.1.0
vyhcbm552rzd   todoapp_db         replicated   1/1        postgres:16-alpine
yn38nsf9zhmz   todoapp_frontend   replicated   2/2        alancor/todo-frontend:v1.1.0   *:80->80/tcp
te7t9fw28gra   todoapp_redis      replicated   1/1        redis:7-alpine
(base) alan@alanC9:~/Documents/maestria/todo-app$
```

```
(base) alan@alanC9:~/Documents/maestria/todo-app$ docker service ps todoapp_api
ID             NAME               IMAGE                          NODE      DESIRED STATE   CURRENT STATE            ERROR     PORTS
ktersdxclcti   todoapp_api.1      alancor/todo-api:v1.1.0        alanC9    Running         Running 3 minutes ago
vmf5akla4lu2    \_ todoapp_api.1  alan/todo-api:v1.0.0           alanC9    Shutdown        Shutdown 3 minutes ago
jt24eej8jpvj   todoapp_api.2      alancor/todo-api:v1.1.0        alanC9    Running         Running 3 minutes ago
qml9nzorio4h    \_ todoapp_api.2  alan/todo-api:v1.0.0           alanC9    Shutdown        Shutdown 3 minutes ago
(base) alan@alanC9:~/Documents/maestria/todo-app$
```

> Clúster Swarm inicializado correctamente, con **réplicas**, **overlay network**, **configs** y **secrets**.
> Cumple el requisito de *Despliegue en Swarm con servicios replicados*.

---

## 6 Despliegue en *Kubernetes (KIND)*

### Crear clúster con puerto NodePort mapeado

```
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
  - role: control-plane
    extraPortMappings:
      - containerPort: 30080
        hostPort: 30080
        protocol: TCP
```

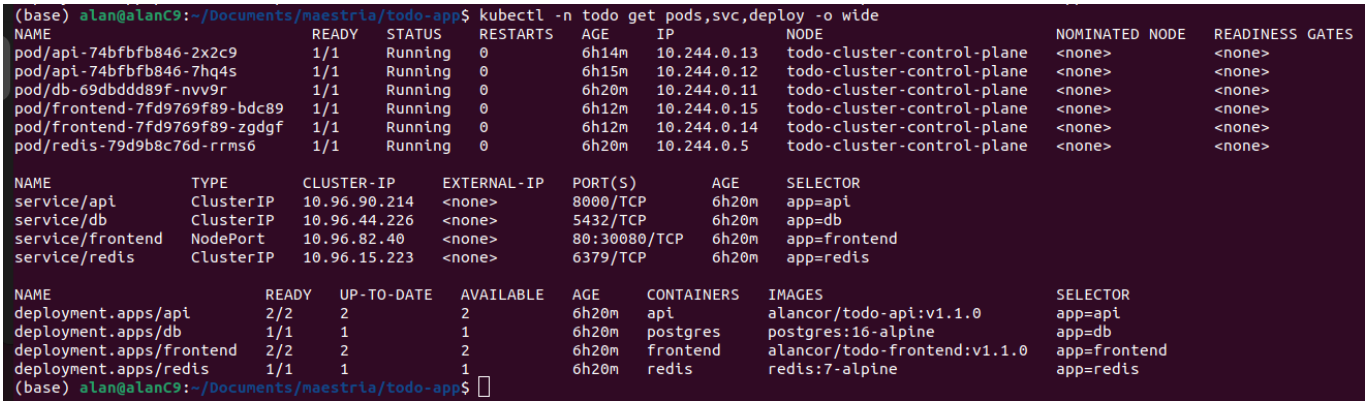
```
kind create cluster --name todo-cluster --config kind-config.yaml
```

### Aplicar manifiestos

```
kubectl apply -f k8s/namespace.yaml
kubectl -n todo apply -f k8s/secret.yaml -f k8s/configmap.yaml \
  -f k8s/db.yaml -f k8s/redis.yaml -f k8s/api.yaml -f k8s/frontend.yaml
kubectl -n todo get pods,svc,deploy -o wide
```

## Resultado observado

Pods y servicios activos en el clúster:

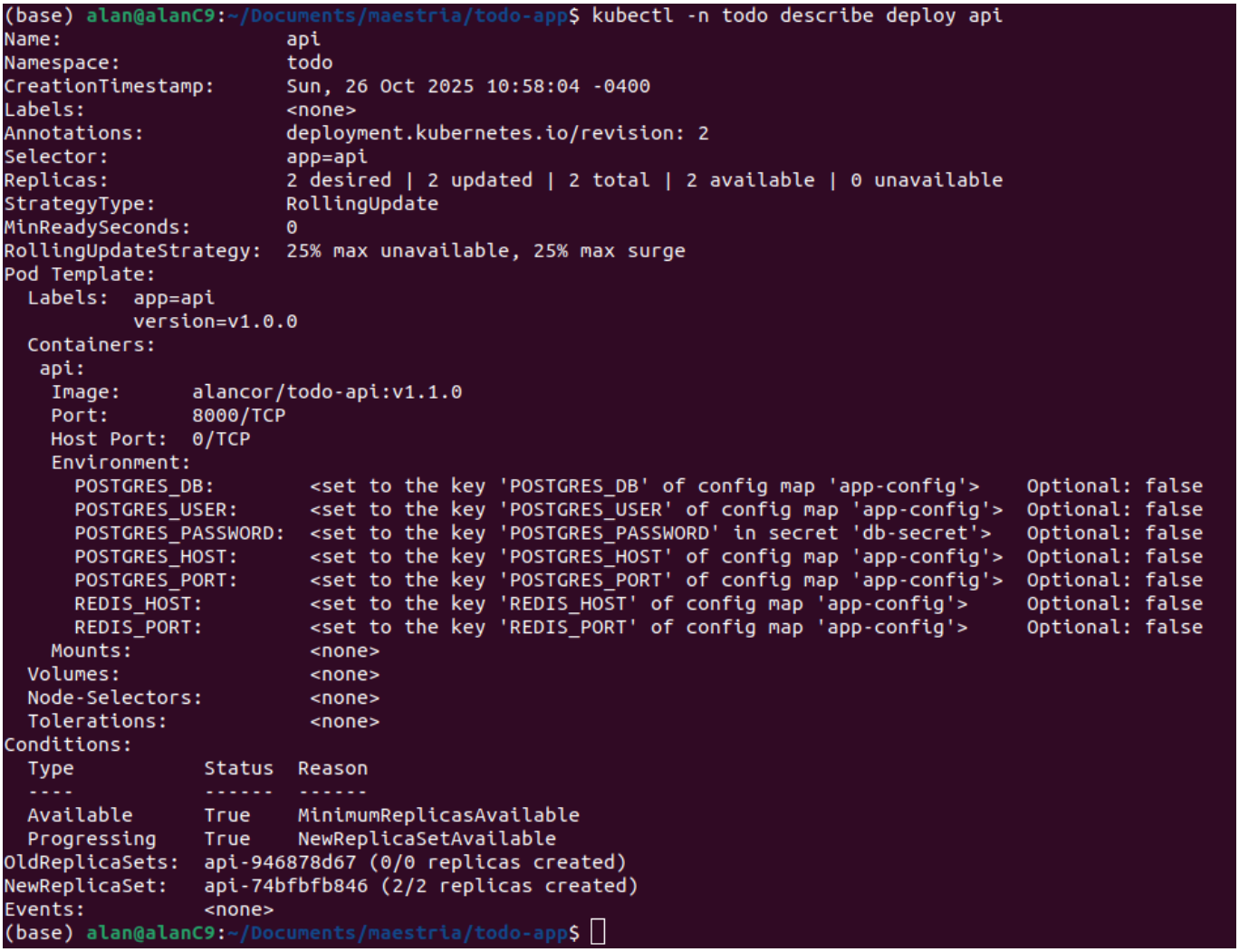


```
(base) alan@alanC9:~/Documents/maestria/todo-app$ kubectl -n todo get pods,svc,deploy -o wide
NAME                            READY   STATUS    RESTARTS   AGE     IP            NODE                       NOMINATED NODE   READINESS GATES
pod/api-74bfbfb846-2x2c9        1/1     Running   0          6h14m   10.244.0.13   todo-cluster-control-plane   <none>           <none>
pod/api-74bfbfb846-7hq4s        1/1     Running   0          6h15m   10.244.0.12   todo-cluster-control-plane   <none>           <none>
pod/db-69dbddd89f-nvv9r         1/1     Running   0          6h20m   10.244.0.11   todo-cluster-control-plane   <none>           <none>
pod/frontend-7fd9769f89-bdc89   1/1     Running   0          6h12m   10.244.0.15   todo-cluster-control-plane   <none>           <none>
pod/frontend-7fd9769f89-zgdgf   1/1     Running   0          6h12m   10.244.0.14   todo-cluster-control-plane   <none>           <none>
pod/redis-79d9b8c76d-rrms6      1/1     Running   0          6h20m   10.244.0.5    todo-cluster-control-plane   <none>           <none>

NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE     SELECTOR
service/api       ClusterIP   10.96.90.214   <none>        8000/TCP       6h20m   app=api
service/db        ClusterIP   10.96.44.226   <none>        5432/TCP       6h20m   app=db
service/frontend  NodePort    10.96.82.40    <none>        80:30080/TCP   6h20m   app=frontend
service/redis     ClusterIP   10.96.15.223   <none>        6379/TCP       6h20m   app=redis

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE     CONTAINERS   IMAGES                          SELECTOR
deployment.apps/api        2/2     2            2           6h20m   api          alancor/todo-api:v1.1.0         app=api
deployment.apps/db         1/1     1            1           6h20m   postgres     postgres:16-alpine              app=db
deployment.apps/frontend   2/2     2            2           6h20m   frontend     alancor/todo-frontend:v1.1.0    app=frontend
deployment.apps/redis      1/1     1            1           6h20m   redis        redis:7-alpine                  app=redis
(base) alan@alanC9:~/Documents/maestria/todo-app$
```

Detalle del despliegue de la API con versión actualizada:



```
(base) alan@alanC9:~/Documents/maestria/todo-app$ kubectl -n todo describe deploy api
Name:                   api
Namespace:              todo
CreationTimestamp:      Sun, 26 Oct 2025 10:58:04 -0400
Labels:                 <none>
Annotations:            deployment.kubernetes.io/revision: 2
Selector:               app=api
Replicas:               2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:           RollingUpdate
MinReadySeconds:        0
RollingUpdateStrategy:  25% max unavailable, 25% max surge
Pod Template:
  Labels:  app=api
           version=v1.0.0
  Containers:
   api:
    Image:      alancor/todo-api:v1.1.0
    Port:       8000/TCP
    Host Port:  0/TCP
    Environment:
      POSTGRES_DB:        <set to the key 'POSTGRES_DB' of config map 'app-config'>    Optional: false
      POSTGRES_USER:      <set to the key 'POSTGRES_USER' of config map 'app-config'>  Optional: false
      POSTGRES_PASSWORD:  <set to the key 'POSTGRES_PASSWORD' in secret 'db-secret'>   Optional: false
      POSTGRES_HOST:      <set to the key 'POSTGRES_HOST' of config map 'app-config'>  Optional: false
      POSTGRES_PORT:      <set to the key 'POSTGRES_PORT' of config map 'app-config'>  Optional: false
      REDIS_HOST:         <set to the key 'REDIS_HOST' of config map 'app-config'>     Optional: false
      REDIS_PORT:         <set to the key 'REDIS_PORT' of config map 'app-config'>     Optional: false
    Mounts:               <none>
  Volumes:                <none>
  Node-Selectors:         <none>
  Tolerations:            <none>
Conditions:
  Type           Status  Reason
  ----           ------  ------
  Available      True    MinimumReplicasAvailable
  Progressing    True    NewReplicaSetAvailable
OldReplicaSets:  api-946878d67 (0/0 replicas created)
NewReplicaSet:   api-74bfbfb846 (2/2 replicas created)
Events:          <none>
(base) alan@alanC9:~/Documents/maestria/todo-app$
```

```
deployment.apps/api        alancor/todo-api:v1.1.0
deployment.apps/frontend   alancor/todo-frontend:v1.1.0
service/frontend NodePort 80:30080/TCP
```

**Acceso**: http://localhost:30080

### Prueba funcional

```
curl http://localhost:30080/api/tasks
```

```
(base) alan@alanC9:~/Documents/maestria/todo-app$ curl http://localhost:30080/api/tasks
[{"done":false,"id":1,"title":"tarea"},{"done":false,"id":2,"title":"Guia latex UCB"},{"done":false,"id":3,"title":"proyecto maestria"}]
(base) alan@alanC9:~/Documents/maestria/todo-app$
```

- Se verificó el correcto funcionamiento del clúster KIND y la exposición de servicios NodePort.
- Cumple Despliegue en Kubernetes (KIND).

---

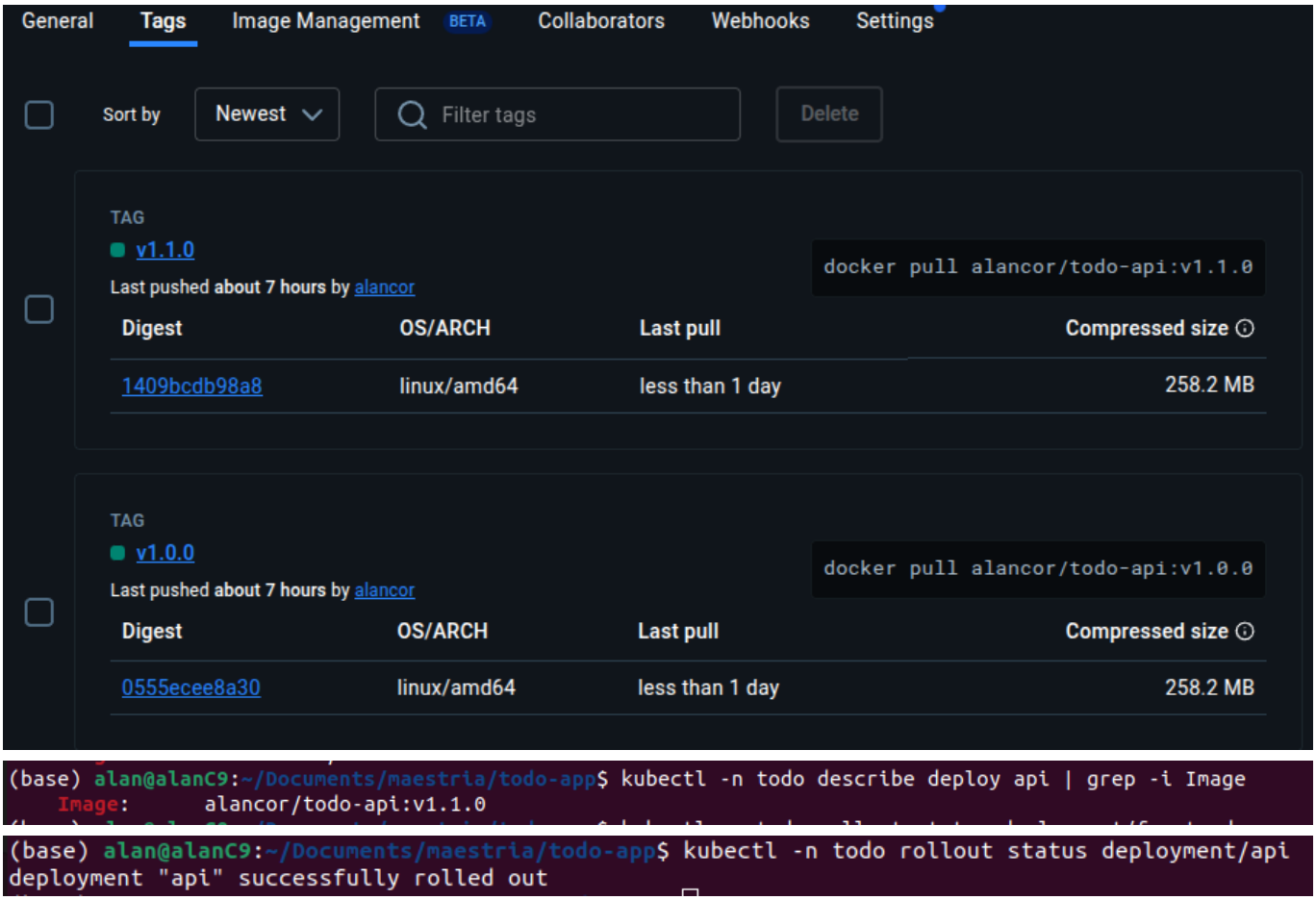
## 7 Versionamiento y *Rolling Update* en K8s

### API (Flask – Backend)

```
# Construcción y push de nueva versión
docker build -t alancor/todo-api:v1.1.0 api
docker push  alancor/todo-api:v1.1.0

# Actualización en el clúster (rolling update)
kubectl -n todo set image deployment/api api=alancor/todo-api:v1.1.0
kubectl -n todo rollout status deployment/api
kubectl -n todo describe deploy api | grep -i Image
```

### Evidencia










```
Image: alancor/todo-api:v1.1.0
```

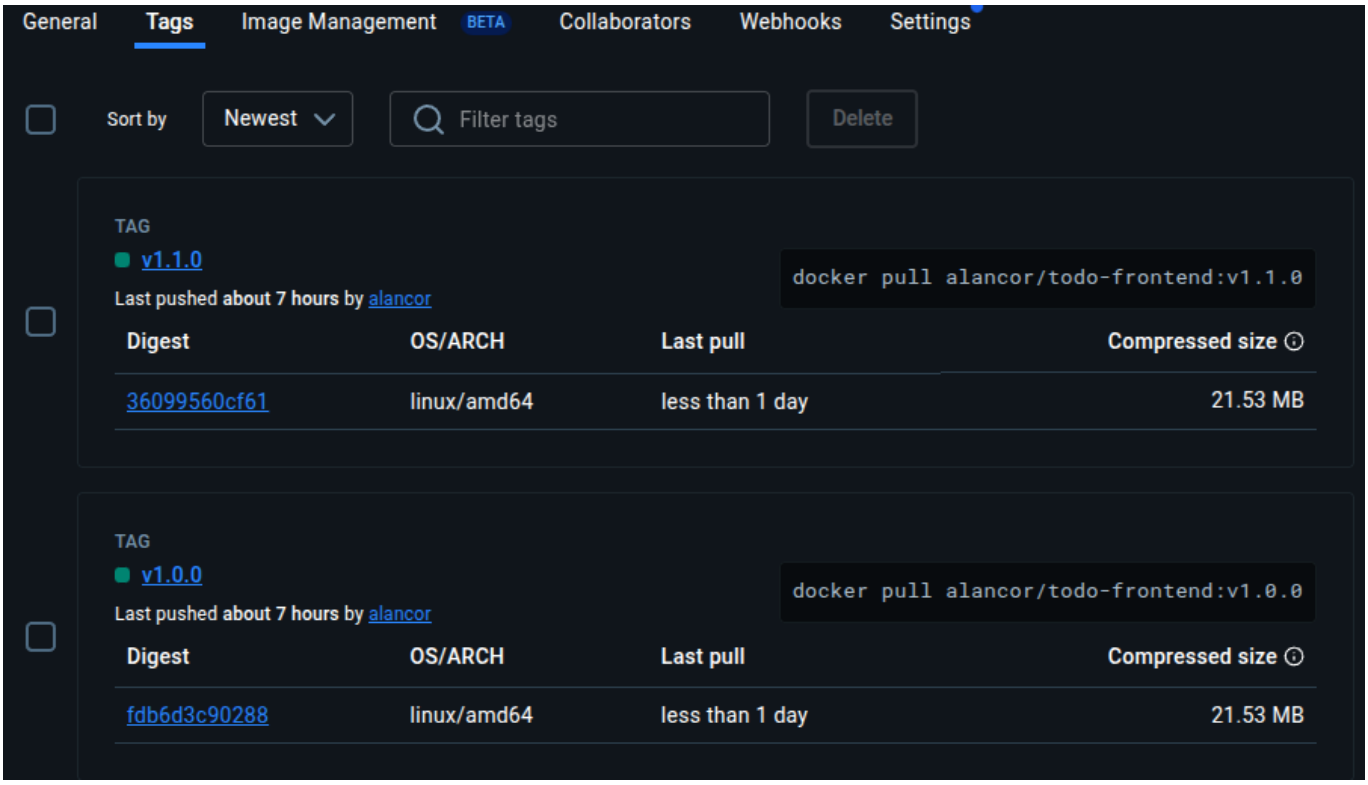
- La API se actualizó correctamente, manteniendo disponibilidad y reflejando la nueva versión

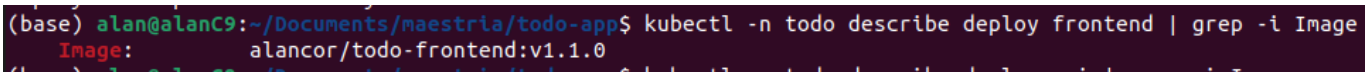
## Frontend (Nginx – Interfaz web)

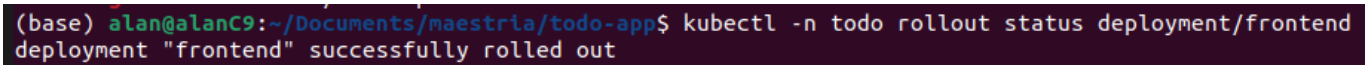
```
# Construcción y push de nueva versión
docker build -t alancor/todo-frontend:v1.1.0 frontend
docker push  alancor/todo-frontend:v1.1.0

# Actualización en el clúster (rolling update)
kubectl -n todo set image deployment/frontend frontend=alancor/todo-
frontend:v1.1.0
kubectl -n todo rollout status deployment/frontend
kubectl -n todo describe deploy frontend | grep -i Image
```

## Evidencia




```
Image: alancor/todo-frontend:v1.1.0
```

- El frontend se actualizó satisfactoriamente, mostrando el encabezado “To-Do v1.1.0” en la interfaz.
- Se realizó sin interrupción del servicio, verificando la correcta aplicación del rolling update en Kubernetes.
- Cumple y supera el requisito de versionamiento mínimo de 2 versiones, aplicando la actualización controlada en dos servicios (API y Frontend) dentro del clúster KIND

| **Antes (v1.0.0)*** | **Después (v1.1.0)** |
| --- | --- |
| *(Interfaz original sin indicador de versión)* |  |



## 8 Verificación final

| Prueba | Comando / URL | Resultado |
|---|---|---|
| Listar pods y servicios | `kubectl -n todo get pods,svc,deploy -o wide` | Todos los pods *Running* en clúster KIND |
| API funcional | `curl http://localhost:30080/api/tasks` | Devuelve JSON con las tareas registradas |
| Frontend actualizado | `http://localhost:30080` | Interfaz visible con título **"To-Do v1.1.0"** |
| Verificar imágenes desplegadas | `kubectl -n todo describe deploy api \| grep -i Image` y `kubectl -n todo describe deploy frontend \| grep -i Image` | Muestran `alancor/todo-api:v1.1.0` y `alancor/todo-frontend:v1.1.0` |
| Swarm status | `docker stack services todoapp` | Servicios replicados y en estado *Running* (`alancor/*:v1.1.0`) |

> Las pruebas confirman el funcionamiento completo de la aplicación en los tres entornos: **Compose**, **Swarm** y **Kubernetes (KIND)**.
> Se validó la comunicación entre servicios, el correcto enrutamiento interno (frontend ↔ api ↔ db ↔ redis) y la persistencia de datos en PostgreSQL.

---

## 9 Evaluación trazada

| Criterio oficial | Evidencia en este proyecto |
|---|---|
| **Arquitectura** | Seis servicios independientes: frontend, api, db, redis, adminer, cadvisor; uso de red interna y variables de entorno |
| **Conteinerización** | Dockerfiles livianos (Alpine), `.dockerignore`, sin credenciales, etiquetas versionadas (`v1.0.0` y `v1.1.0`) |
| **Compose / Swarm** | `docker-compose.yml` y `stack-deploy.yml` con réplicas, configs y secrets; validado con `docker stack services` |
| **Kubernetes** | Manifiestos YAML completos (Namespace, Secret, ConfigMap, PVC, Deployments, Services NodePort); clúster KIND funcional |
| **Documentación** | Este README con pasos reproducibles, evidencias de ejecución y capturas de verificación |

> Se cumplen **todos los criterios de evaluación** establecidos en la guía del examen final, incluyendo versionamiento, orquestación y documentación técnica reproducible.

---

## Conclusión

El proyecto demuestra la migración completa **del desarrollo local a la orquestación en clúster**, manteniendo la misma arquitectura modular y escalable.
Se lograron todos los objetivos específicos del examen:

- **Imágenes personalizadas y públicas** en Docker Hub (`alancor/*`)
- **Versionamiento mínimo y rolling updates** aplicados en **API** y **Frontend**
- **Despliegue funcional** en *Docker Compose*, *Docker Swarm* y *Kubernetes (KIND)*
- **Documentación integral y reproducible**, evidenciando el ciclo completo de containerización y orquestación
- **Código fuente** publicados en el repositorio público de GitHub: AlanCornejoQ/todo-app