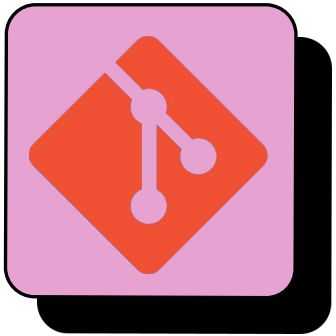
PRESENTACIÓN DE TPS PRÁCTICOS
ING DE SOFTWARE III

De Marcos - Strumia

ÍNDICE



GIT

FORK, CLONE E IDENTIDAD

FORK:

Copiamos el repo a nuestro GitHub para trabajar de forma independiente.

CLONE:

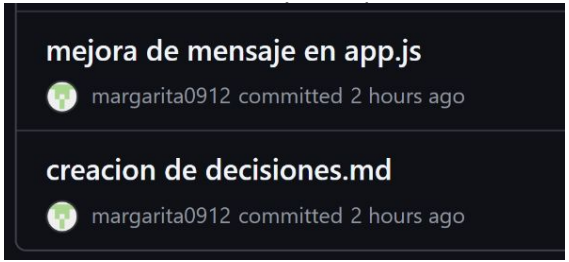
Lo bajamos a nuestra máquina usando: git clone

IDENTIDAD:

Configuramos nombre y mail para que nuestros commits sean identificables con:
git config user.name / git config user.email

RAMA FEATURE Y COMMITS

- **Rama feature** creada desde main para trabajar de forma aislada sin romper la rama estable.
- **Commits atómicos** (cambios pequeños y claros):
    - "creacion de decisiones.md"
    - "mejora de mensaje en app.js"
- **Justificación**: Git Flow
    - Branch main siempre estable
    - Branch feature para nuevas funcionalidades.



RAMA HOTFIX Y PULL REQUESTS

BRANCH HOTFIX

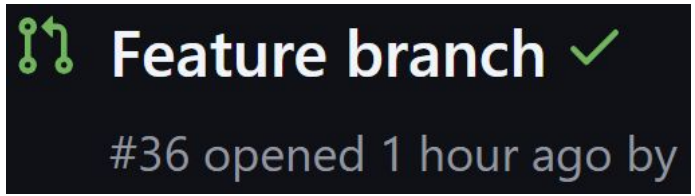
- **Simulamos un error** en main.
- **Creamos rama hotfix** y corregimos rápido.
- **Merge en main** de hotfix con --no-ff para registrar el fix.

```
git merge --no-ff hotfix -m ""
```

- **Justificación:** esto se hace para aislar y resolver errores sin afectar otras ramas.

PULL REQUESTS

- **Subimos la rama feature**
- **Integramos en la rama main** desde GitHub
- Colocamos **título y descripción** para dejar claros los cambios

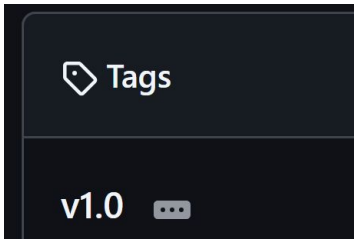


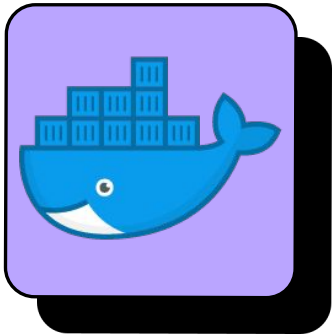
VERSIONAMOS EL TRABAJO

- Desde la rama main
  - En la terminal:

```
git tag v1.0 -m "Versión estable inicial v1.0"
git push origin v1.0
```

- Allí creamos el tag y lo subimos
- Se usó la convención SemVer





DOCKER

DOCKERFILE E IMAGEN

- **Base:** punto de partida de la imagen, FROM golang:1.23 trae el compilador y las librerías necesarias de Go
- **RUN go build -o server . :** compila el código Go y genera un binario llamado server
- **EXPOSE 8081:** documenta puerto de la app
- **CMD:** ejecuta automáticamente ./server y lo deja corriendo en el puerto 8081

```
FROM golang:1.23

WORKDIR /app
COPY go.mod go.sum ./
RUN go mod download

COPY . .
RUN go build -o server .

EXPOSE 8081
CMD ["./server"]
```

ESTRATEGIA QA Y PROD

- **Misma imagen, distinta configuración**

- **QA:** puerto 8081, DB 3307, modo DEBUG
- **PROD:** puerto 8082, DB 3308, modo INFO
- **Variables de entorno:** controlan puertos, logs y credenciales
- **Beneficio:** entornos reproducibles y seguros

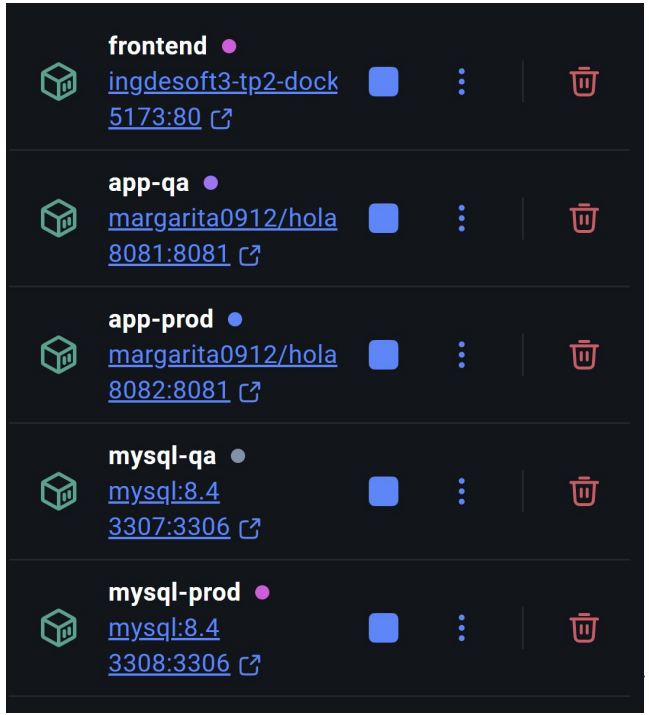
DOCKER COMPOSE

Servicios definidos:

- mysql-qa / mysql-prod
- app-qa / app-prod
- frontend (5173)

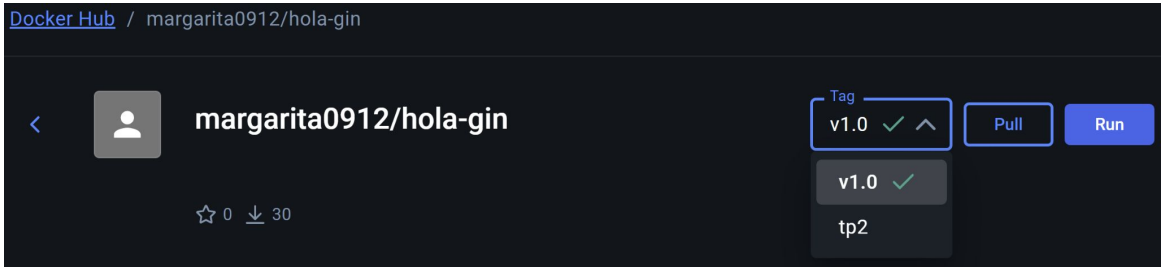
Buenas prácticas:

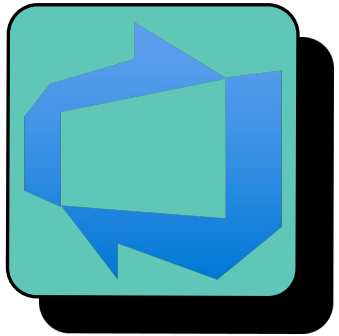
- Redes aisladas (qa/prod)
- Healthchecks para dependencias
- Puertos únicos → sin conflictos



VERSIONADO Y PUBLICACIÓN

- Imagen subida a Docker Hub
- Versionado
  - tp2 = versión inicial
  - v1.0.0 = primera versión estable
- Control de cambios claro y profesional





AZURE DEVOPS

ORGANIZACIÓN Y PROYECTO

01

CREAR ORGANIZACIÓN

En estados unidos

02

CREAR PROYECTO

Privado
Control de versiones GIT
Metodología SCRUM

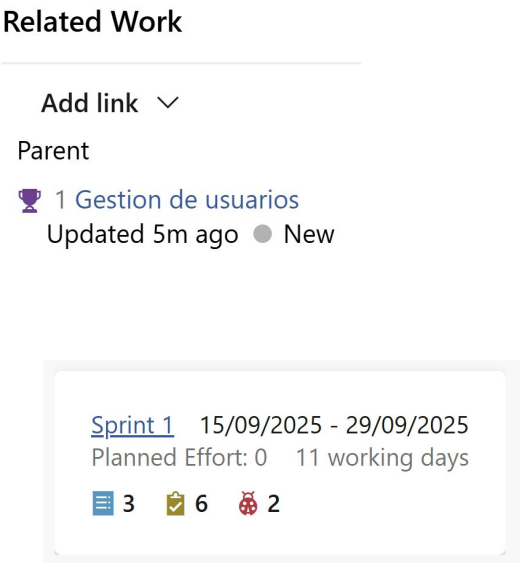
GESTIÓN DE TRABAJO

en Azure Boards

Jerarquía de trabajo:

- **Epic:** Gestión de usuarios.
- **PBIs:** crear, editar y eliminar usuarios.
- **Tasks:** formularios + endpoints backend.
- **Bugs creados:**
  - No valida campos obligatorios en alta.
  - Error 500 al eliminar usuario inexistente.

Sprint 1: 15/09 – 29/09/2025 → incluye todos los PBIs, Tasks y Bugs.



CONTROL DE VERSIONES

Repositorio creado con rama principal main.

Políticas en main:

- Pull Request obligatorio para merge.
- Al menos 1 revisor distinto al autor.

Ramas feature:

- feature/CreateUser, feature/SearchUsers.

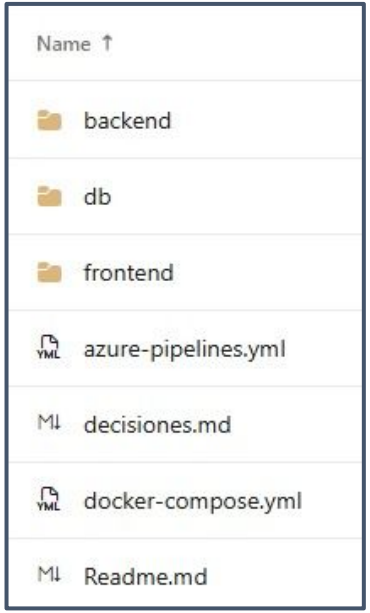
Pull Requests:

- Creados desde features → main, pendientes de aprobación (faltan revisores).

AZURE PIPELINES

INSTALACIÓN Y SETUP INICIAL

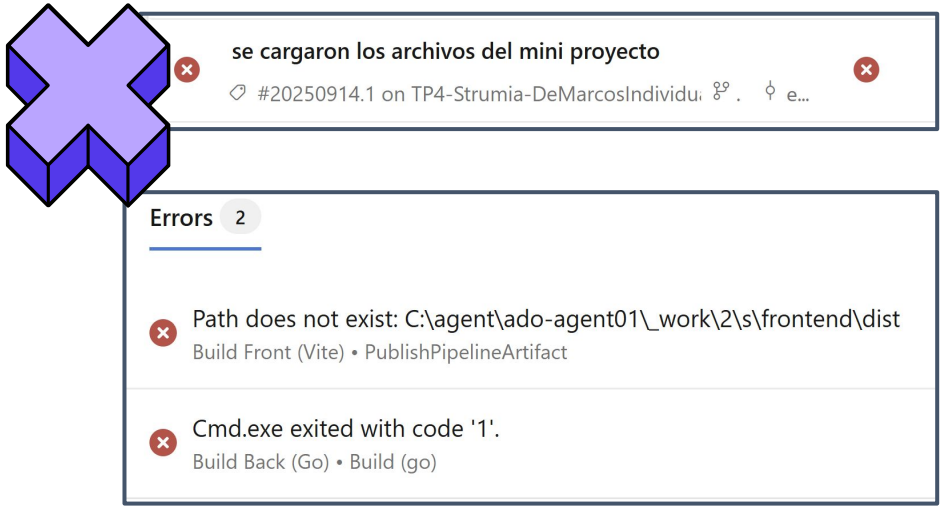
- Instalación del **agente** para ejecutar pipelines.
- Creación de **Access Token** con scope Agent Pools (Read & Manage).
- Creación de un **Agent Pool** para pruebas.
- **Pipeline simple inicial →** verificación de autorización.
- **App web cargada en repo con estructura:**
  /frontend, /backend, azure-pipelines.yml.



PRIMER PIPELINE CREADO

Ambos fallaron inicialmente:

- Front: faltaba frontend/dist.
- Back: agente sin Go instalado.

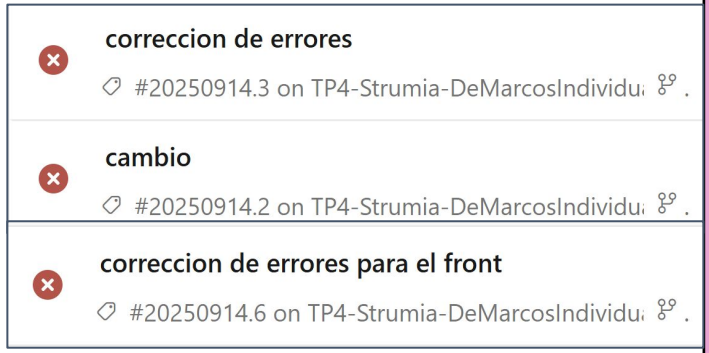


SOLUCIÓN DE PROBLEMAS

FRONTEND

- Ajustamos la configuración para que el build genere la carpeta correcta (dist).

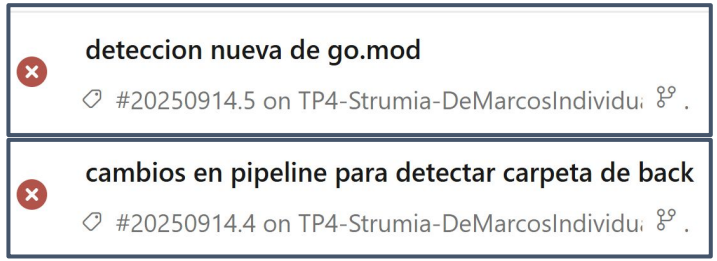
- Se probó el job con npm install + build y finalmente produjo el artefacto del front (archivos estáticos listos para usarse).



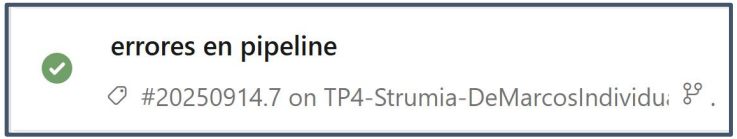
SOLUCIÓN DE PROBLEMAS

BACKEND

- Se corrigió la ruta del proyecto, ya que el código no estaba donde el pipeline lo esperaba.
- Se agregó un paso de detección automática para ubicar la carpeta del back.
- Se solucionaron problemas de caché/configuración que impedían compilar.
- Finalmente, el pipeline construyó el ejecutable del backend (server).



PIPELINE FINAL ESTABLE



Frontend:

- Configurado con Node 18.
- Se agregó caché para acelerar instalaciones.

Backend:

- Se agregaron validaciones para asegurar que todo esté en su lugar antes de compilar.
- Se corrigieron rutas y configuraciones.

Resultado Final

- Un pipeline estable con jobs independientes que pueden correr en paralelo.
- Publica artefactos front y back listos

¡GRACIAS!