

Informe del Proyecto: **TalentNest**

Red Social Distribuida

Equipo: *Diego y Pablo*

Asignatura: Sistemas Distribuidos

30 de noviembre 11:59:59 pm

Índice

| | |
|--|----------|
| 1. Resumen | 2 |
| 2. Objetivos del sistema | 2 |
| 3. Arquitectura y diseño del sistema | 2 |
| 4. Organización del sistema distribuido | 2 |
| 4.1. Servicios principales | 2 |
| 5. Roles del sistema | 2 |
| 6. Procesos y servicios | 3 |
| 6.1. Tipos de procesos | 3 |
| 6.2. Agrupación de procesos | 3 |
| 7. Patrón de ejecución y desempeño | 3 |
| 7.1. Concurrencia | 3 |
| 8. Comunicación entre componentes | 3 |
| 8.1. Tipos de comunicación | 3 |
| 8.2. Acceso exclusivo y condiciones de carrera | 3 |
| 8.3. Toma de decisiones distribuidas | 4 |
| 9. Nombrado y localización | 4 |
| 9.1. Ubicación y localización | 4 |
| 10. Consistencia y replicación | 4 |
| 10.1. Replicación de datos | 4 |
| 10.2. Confiabilidad tras actualizaciones | 4 |
| 11. Tolerancia a fallas | 4 |
| 11.1. Respuesta a errores | 4 |
| 11.2. Fallos parciales | 5 |
| 12. Seguridad | 5 |

1. Resumen

Este documento describe el diseño, la organización, los procesos y las decisiones arquitectónicas del sistema distribuido **TalentNest**, una red social distribuida diseñada para compartir perfiles profesionales, proyectos y oportunidades. El documento cubre aspectos de arquitectura, comunicación, coordinación, consistencia, tolerancia a fallos y seguridad.

2. Objetivos del sistema

- Proveer una plataforma escalable para compartir perfiles y publicaciones, y mensajes dentro de las publicaciones inspirado en Linkedin.
- Diseño distribuido que permita replicación de datos y alta disponibilidad.
- Mecanismos de autenticación, autorización y comunicación segura entre componentes.

3. Arquitectura y diseño del sistema

TalentNest sigue una arquitectura cliente-servidor desplegada sobre contenedores de Docker. Cada servicio es responsable de una función (perfil de usuario, publicaciones, autenticación, notificaciones).

4. Organización del sistema distribuido

4.1. Servicios principales

1. **Frontend** (app web / SPA): Interfaz de usuario.
2. **Auth Service**: Manejo de identidades, tokens JWT.
3. **User Service**: CRUD de perfiles y relaciones.
4. **Post Service**: Generación y edición de posts.
5. **Notification Service**: Encolado y entrega de notificaciones.
6. **Database(s)**: Almacenamiento de datos primarios.

5. Roles del sistema

| Rol | Descripción |
|----------------------|--|
| Auth Service | Emisión y verificación de tokens; gestión de sesiones. |
| User Service | Manejo de perfiles, contactos, configuración. |
| Post Service | Agregación de posts, edición y eliminación. |
| Notification Service | Encolado (pub/sub) y entrega de notificaciones. |

Cuadro 1: Roles principales en TalentNest

6. Procesos y servicios

6.1. Tipos de procesos

- **Procesos sin estado** (stateless): Frontend.
- **Procesos con estado** (stateful): Databases.
- **Workers** asíncronos: Backend.

6.2. Agrupación de procesos

Todos los servicios pertenecen a un mismo nodo Backend, junto con la base de datos embebida, y otro nodo para el proceso del cliente que sería el Frontend

7. Patrón de ejecución y desempeño

7.1. Concurrencia

El sistema utiliza concurrencia basada en Go, aprovechando goroutines y channels para ejecutar tareas lightweight sin bloquear hilos del sistema operativo. Esto permite manejar comunicación entre servicios y paralelización de tareas internas de forma eficiente.

Las goroutines permiten lanzar tareas concurrentes de bajo costo, mientras que los channels ofrecen un mecanismo seguro de comunicación y sincronización.

8. Comunicación entre componentes

8.1. Tipos de comunicación

- **Cliente-Servidor:** HTTP RESTful API (JSON) entre frontend y API Gateway.
- **Servidor-Servidor:** Planteamos una arquitectura Master-Slave entre los nodos del backend, el nodo líder replica información a sus seguidores vía RPC.

8.2. Acceso exclusivo y condiciones de carrera

Permiso de escritura exclusivamente en el nodo líder y permiso de lectura en todos los nodos, además de balanceo de carga por parte del DNS de docker (Round-Robin) para evitar servidores cargados de requests.

8.3. Toma de decisiones distribuidas

Al aplicar la arquitectura de master-slave, la toma de desiciones las realiza el nodo lider, en este caso la elección del lider se hace mediante un algoritmo personalizado de elección de lider donde se escoge el lider de acuerdo al tamaño del numero de su ip en la red.

9. Nombrado y localización

Cada servicio y dato tiene identificadores globales:

- Servicios: nombres DNS dentro de la red Docker (`backend:5000`).

9.1. Ubicación y localización

La localización se resuelve por:

- DNS interno.
- como método alternativo usamos ip-cache, en caso de caída del DNS

10. Consistencia y replicación

El sistema no es consistente al existir particiones, ya que se priorizaron la Tolerancia a Particiones y la Disponibilidad en este Sistema Distribuido.

10.1. Replicación de datos

Workers del Backend: replicación maestro-esclavo (1 primario, varios secundarios) o replicación multi-Master si se requiere escritura desde varios puntos, como puede ser en el caso de existir diferentes particiones.

10.2. Confiabilidad tras actualizaciones

Tras actualizaciones del sistema, este se mantiene confiable, siempre que no esté particionado, pero en caso de fusión de las particiones se trabaja con un sistema de colas con prioridad en dependencia de las acciones de cada partición para volver a garantizar consistencia en todo el sistema.

11. Tolerancia a fallas

11.1. Respuesta a errores

- Replicación de los workers en casos de caídas de los nodos del sistema
- Timeouts razonables y vigilancia (health checks) para reiniciar contenedores fallidos.

11.2. Fallos parciales

- Nodos caídos temporalmente: reconexión automática y recuperación del estado a partir de las colas.
- Nodos nuevos: descubrimiento automático vía DNS y replicación de datos.

12. Seguridad

- Nada
- está
- seguro