Documentación
Proyecto final

Ingeniería de software

Julio Daniel Sincal Aroche

Jesus Fernando Ruiz Castañaza

David Roberto Recinos Orellana

31 de mayo de 2025

Proyecto Task Manager en Kubernetes

Introducción

Este documento describe la arquitectura, configuración y despliegue de la aplicación Task Manager en un clúster de Kubernetes. Incluye detalles sobre los componentes principales, la estructura de directorios, los manifiestos de Kubernetes y los pasos necesarios para implementar y acceder a la aplicación.

Descripción de la Arquitectura

La aplicación Task Manager está compuesta por los siguientes componentes desplegados en Kubernetes:

- Frontend: Aplicación React servida por Nginx.
- API: Servicio Node.js/Express que maneja la lógica de negocio y operaciones CRUD.
- Base de datos: MongoDB para almacenamiento de datos.
- Monitoring: Prometheus y Grafana para recolección y visualización de métricas.
- Ingress NGINX: Maneja el enrutamiento de tráfico externo a los servicios Frontend, API, Prometheus y Grafana.
- Secrets y ConfigMaps: Para gestionar credenciales y configuraciones.
- Persistent Volume Claims: Para almacenamiento persistente de MongoDB, Prometheus y Grafana.

-Keycloak para la autenticación

Estructura de Directorios

La raíz del proyecto contiene los siguientes componentes y carpetas:

- k8s/: Directorio principal con las configuraciones de Kubernetes organizadas por funcionalidad.
  - frontend/: Manifiestos para el despliegue del Frontend.
  - api/: Manifiestos para el despliegue de la API Node.js/Express.
  - mongo/: Manifiestos para la base de datos MongoDB (Deployment, Service, PVC, Secret).
  - monitoring/: Manifiestos para Prometheus y Grafana (ConfigMaps, Deployments, Services, Ingress, PVC).
- .gitignore: Archivo para ignorar archivos en el repositorio.
- graph.json: Código para la creación del dashboard

- README.md: Archivo de documentación básica.

Funciones

Aplicación funcional desplegada en Kubernetes

La aplicación Task Manager se ha empaquetado en contenedores Docker para frontend (React) y backend (Node.js/Express), y se ha desplegado en un clúster de Kubernetes. Esto permite una orquestación escalable y gestionada de los componentes.

Backend y/o frontend en contenedores

El backend y el frontend han sido contenedorizados usando imágenes de Docker. El frontend usa Node.js y Nginx para servir la aplicación React, mientras que el backend ejecuta Node.js/Express. Estas imágenes se almacenan en un registro y se usan en los manifiestos para crear Pods.

Uso de Deployment, Service y Ingress

Los manifiestos de Kubernetes incluyen recursos Deployment para gestionar réplicas y actualizaciones sin downtime. Los Services exponen los Pods internamente, y un Ingress (NGINX Ingress Controller) redirige el tráfico HTTP al frontend y la API según el host y la ruta.

Uso de Helm o manifiestos YAML bien estructurados

Se han creado manifiestos YAML para cada componente organizados en carpetas lógicas. Opcionalmente, se puede usar un chart de Helm para parametrizar variables como réplicas, configuraciones y dependencias.

Repositorio en Git con historial de cambios

El proyecto se almacena en un repositorio GitHub que contiene todo el código fuente, Dockerfiles, manifiestos de Kubernetes y archivos de configuración. El historial muestra los commits y cambios a lo largo del desarrollo.

Uso de secrets/config maps de forma segura

Se utilizan Secrets de Kubernetes para almacenar credenciales de la base de datos y tokens de la aplicación. ConfigMaps contienen configuraciones no sensibles como variables de entorno y parámetros. De esta manera, no se incluyen datos sensibles en el código fuente.

Implementación de monitorización

Prometheus se despliega como un Deployment que recolecta métricas de la API expuestas en /metrics. Grafana se despliega para visualizar los datos de Prometheus usando dashboards personalizados.

Uso de volúmenes persistentes (PVCs)

Los Pods de MongoDB, Prometheus y Grafana usan PersistentVolumeClaims para almacenar datos de forma permanente, asegurando que la información no se pierda al reiniciar los Pods.

Implementación de autenticación en la app

Se incluye autenticación con Keycloak para proteger las rutas.

Alta disponibilidad: uso de réplicas y readiness/liveness probes

Los Deployments de frontend y backend usan al menos 2 réplicas. Se configuran readiness y liveness probes para garantizar que Kubernetes sólo dirija tráfico a Pods saludables.

Diagramas del sistema

A continuación, se presentan los diagramas de arquitectura y flujo del proyecto.

Diagrama de Arquitectura

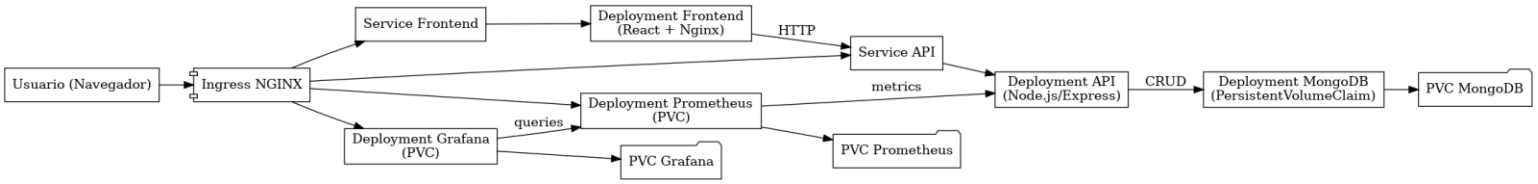
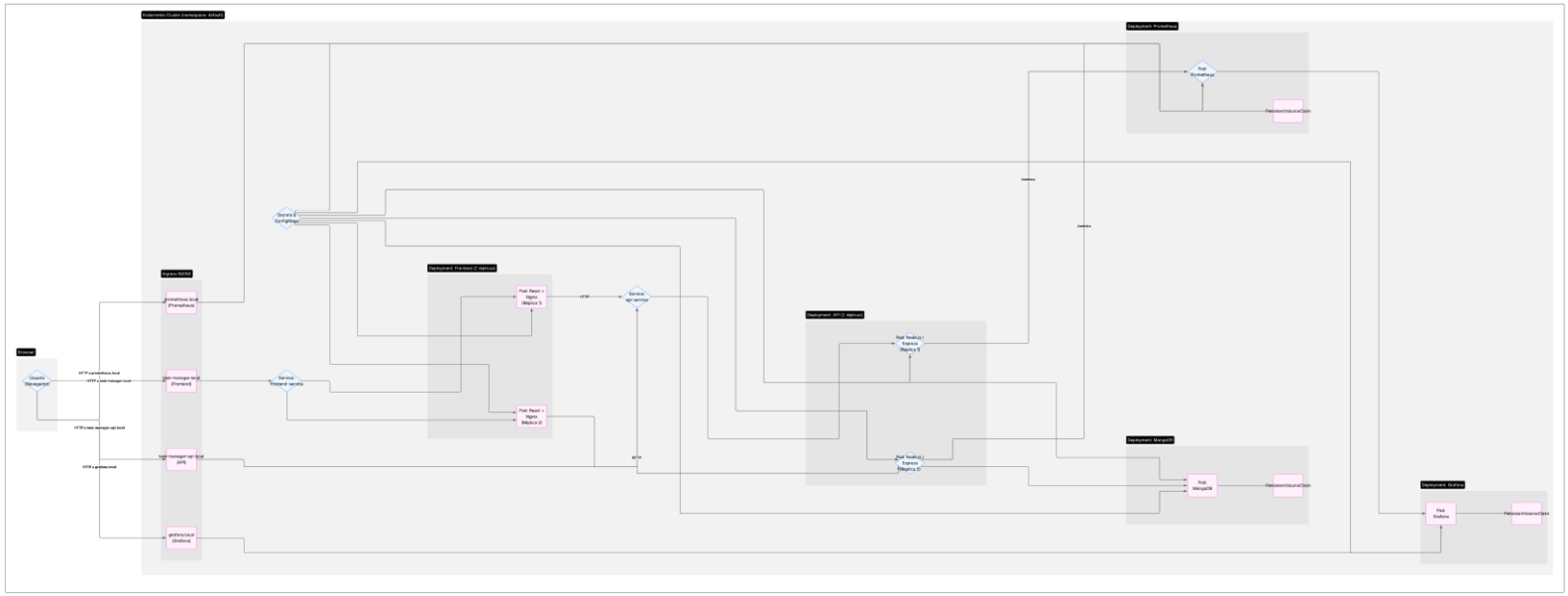


Diagrama de Arquitectura



https://lucid.app/lucidchart/50709429-9243-46ea-83f3-094175e705c4/edit?viewport_loc=59%2C1263%2C3535%2C1539%2C0_0&invitationId=inv_89310d0e-76b9-434e-91c2-b58b5cb5ae51

Diagrama de Flujo