Proyecto de Cloud Computing

Despliegue de una aplicación distribuida

Table of Contents

INTRODUCCIÓN

El proyecto planteado por la asignatura consiste en realizar el diseño e implementación de un sistema distribuido utilizando diferentes tecnologías tales como Docker Compose, Kafka, MINio, Python y Django como lenguajes y framework para el desarrollo y PostgreSQL para la base de datos.

La aplicación desarrollada consiste en descargar ficheros de repositorios git y almacenarlos en el ssistema de datos distribuido MINio.

Se desarrollaron los siguientes componentes:

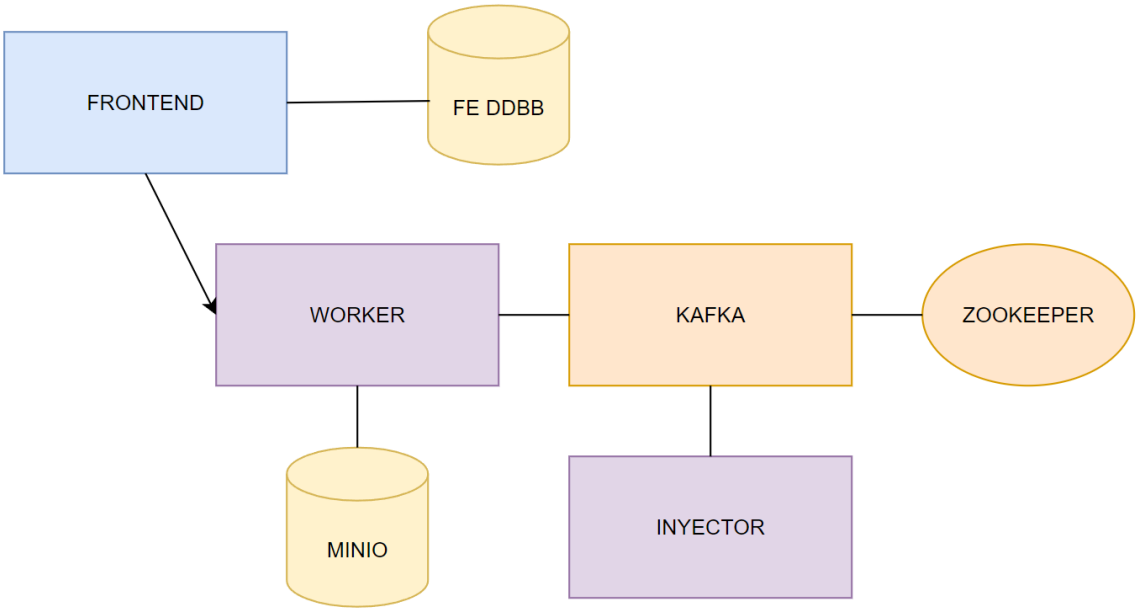
- FrontEnd.
- BD para el FrontEnd
- Injector
- Worker

Se configuran los siguientes componentes:

- Kafka
- Zookeeper
- Keycloak

DESCRIPCIÓN DE LA ARQUITECTURA

El diseño de la aplicación es el siguiente:



El enfoque es que desde el Frotend se suban los enlances de los repositorios git que se desean guardar, se usará además un worker que cuenta con dos funciones, una para guardar el archivo de que contiene los datos del repositorio en una carpeta propia dentro del worker y una adicional que se encargará de enviar el fichero a MINIO.

ELEMENTOS FUNCIONALES

Cada uno de los elementos funcionales son directamente contenedores que están funcionando haciendo uso de Docker para poder ser ejecutados e iniciados al mismo tiempo.

FRONT END

El FrontEnd se encarga de recibir las peticiones del usuario mediante la interfaz gráfica.

Este componente se encarga de guardar en su base de datos propia, la información que ingresa el usuario así como el envío de mensajes al componente Kafka mediante el topic “pendientes”.

KAFKA

Este componente se encarga de escuchar los mensajes que recibe del FrontEnd en el topic “pendientes”. En esta solución, el componente Kafka está cofigura como un nodo Sencillo debido a que el alcance de la solución no requiere de una mayor dispobinilidad.

ZOOKEEPER

Este componente se encarga de la gestión del componente Kafka y sus metadatos. Para nuestro caso de uso, solo se configuró este componente.

INYECTOR

El inyector es un componente importante ya que es este componente el encargado de escuhar en el topic de Kafka por nuevos mensajes

WORKER

El worker realiza las dos tareas más importantes de la aplicación.

1. Se encarga de realizar la descarga de repositorios de la plataforma GitHub.

   La descarga se realiza en una carpeta local del contenedor.

2. Se encarga de realizar la subida de los archivos a el componente MINio.

   En este caso, se toman los archivos de la carpeta temporal en el contenedor y secrea un bucket por cada repositorio. Una vez subido el repositorio, se elimina la carpeta dentro del contenedor

MINio

Componente que se encarga de almacenar los repositorios que el usuario ingresa en el FrontEnd. Se almacena cada repositorio en su propio bucket

BASES DE DATOS

BD - FRONTEND

La base de datos del FrontEnd consiste de una tabla que contiene los siguientes campos:

- repo_url: Almacena la url del repsitorio
- creation_timestamp: Campo que almacena fecha y hora en la que se crea el mensaje en Kafka
- worker_start_timestamp: Momento en el que el worker inicia sus tareas
- worker_finishing_timestamp: Momento en el que el worker termina sus tareas
- inyector_timestamp: Momento en el que el inyector recibe la petición de Kafka

BD - KEYCLOAK

Esta nbase de datos es de uso exclusivo del componente Keycloak para guardar toda la información requerida.

PUNTOS A TOMAR EN CUENTA / NOTAS SOBRE EL DESARROLLO

En esta sección se describirá más funcionalidad implementada relativa a los componentes anteriores.

Para iniciar la solución, basta con ubicarse en el folder raíz del proyecto, en este caso “CC/CC”. Es en este punto donde se involucra un Makefile, por lo que para iniciar la aplicación, basta con escribir en la terminal: “make build”.

Se recomienda una espera de alrededor de 5 minutos para que la aplicación complete su inicio

LIBRERÍAS EXTERNAS

A nivel general y gracias a las extensibilidad del lenguaje Python, se hizo el uso de diversas librerías que permtieron agregar funcionalidades miscelaneas a la solución. Como por ejemplo, el uso de la librería docker de Python pra poder obtener información de manera programática de un contenedor en específico.

Además, de la librería FastAPI para lograr crear un endpoint del lado del compnente Worker de manera que cada vez que sea necesario, se puede invocar a este componente para realizar el trabajo respectivo.

EJEMPLO

En determinado momento, se volvió necesario conocer la dirección IP de los contenedores para poder comunicarse directamente con ellos y sin fallos. Para esto, se programó la funcioinalidad que permite, dado el nombre del contenedor, obtener su dirección IP, de modo que se facilitara la comunicación con el contenedor

DOCKER VOLUMES

Se utilizaron Docker Volumes en aquellos componentes que requerían de un desarrollo continuo, es decir, se volvía ineficiente para probar una funcionalidad, eliminar contenedores para volverlos a crear e iniciar.

Los Docker Volumes evitaron esta situación ya que permitían compartir información entre la máquina host y el contenedor

CONCLUSIONES Y TRABAJO FUTURO

A continuación, se enumeran futuras mejoras a la solución

Componente Keycloak

- Implementar el login utilizando Keycloak como IAM
- Implementar el login utilizando otros medios de inicio de sesión (Github por ejemplo)

1. Agregar automatización para lograr que al inicial el contenedor de Keycloak, usuarios, realm, permisos y otros se creen al iniciar el contenedor

Componente Kafka

- Creación automática de Topics al crear el contenedor
- Configurar el sistema para el uso de un un número mayor de brokers
- Creación de topics “en_proceso” y “completados”

Componente Inyector

- Agregar funcionalidad para permitir el desacargo masivo de repositorios, es decir, el inyector, por medio del API de GitHUb, podría descargar los 10 repositorios en tendencia
- Actualizar la BD con los timestamps requeridos

Componente Worker

- Actualizar la BD con los timestamps requeridos
- Desarrollo de un API con más endpoints y a la vez, más robusta
- Descargar repositorios Github privados y ssh, ya que actualmente solo funciona con repositorios cuya dirección es http y que son públicos

FrontEnd

- Modificar la interfaz de usuario para mostrar la lista de trabajos pendientes, en proceso y completados
- Modificar la UI para poder mostrar las estadísticas de cada repositorio y de la solución en general

General

- Desarrollar más APIs para permitir mayor flexibilidad y escalabilidad al sistema