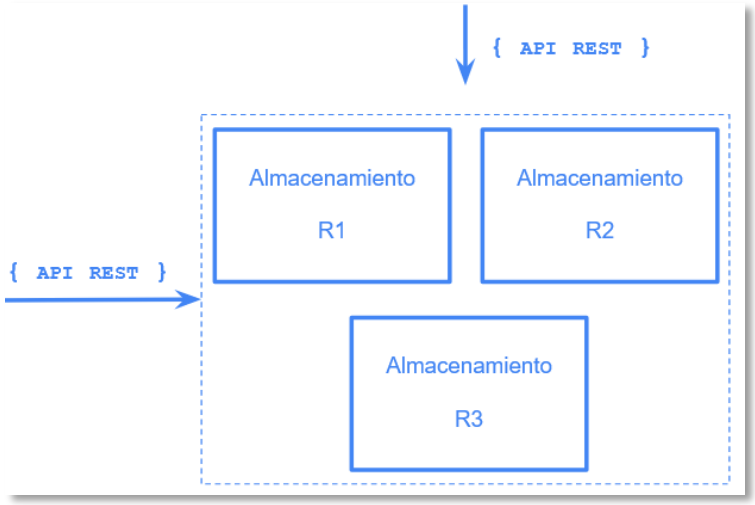
Docente:	**Edwin Boza G.**	**Periodo:**	**PEL3**	**Año:**	**2023**
Materia:	**Sistemas Distribuidos**				
Actividad:	**Tarea 03: Diseño e implementación básica del módulo de almacenamiento**				

Descripción general

Con esta tarea se busca formalizar el diseño del módulo de almacenamiento del proyecto integrador, así como reportar el avance realizado del proyecto.



Indicaciones:

1. *Utilice este documento como formato para realizar la tarea. Obtenga una copia del documento compartido, compártalo con los miembros del grupo y el docente y responda en cada una de las secciones indicadas a continuación.*
2. *En cada sección, puede eliminar la descripción, que se encuentra escrita con cursiva.*
3. *Se recomienda que este documento se lo realice y entregue lo antes posible para poder recibir retroalimentación respecto al diseño planteado.*
4. *La fecha de entrega de este documento no implica que haya retraso en la fecha de entrega del proyecto. El proyecto debe ser entregado en la 7ma semana de clases (Julio 12).*
5. *Si está desarrollando un tema propio, por favor agregue una sección inicial en la que se describa el alcance del módulo de almacenamiento, similar a como se describe el módulo de almacenamiento en el proyecto general.*
6. *Debido a que el proyecto ya fue enviado hace varias semanas y se debe estar trabajando en él, no se justifica el retraso en la entrega de esta Tarea 03. Solo se evaluarán los documentos enviados hasta la fecha indicada en la actividad. Recuerde que es preferible enviar una tarea incompleta que no enviar tarea.*
7. *El entregable es la versión en PDF de este documento.*
8. *Recuerde que todos los integrantes del grupo deben subir el mismo archivo PDF en su aula virtual.*

Grupo

- *DANNY MORA MORÁN*
- *CHRISTIAN FRANCISCO GOMEZ*
- *CRISTOPHER LARA*
- *JORGE RIGOBERTO VILLANUEVA*

Docente:	Edwin Boza G.	Periodo:	PEL3	Año:	2023
Materia:	Sistemas Distribuidos				
Actividad:	Tarea 03: Diseño e implementación básica del módulo de almacenamiento				

Diseño del módulo de almacenamiento

Descripción de la API REST (Almacenamiento)

SERVICIO / OPERACIÓN	*ENDPOINT*	*DATOS QUE RECIBE*	*OPERACIÓN*	*VALORES DE RETORNO*
Verificación de formulario Duplicado	*/API/VerificaFormularioDuplicado/{id}*	*{id}*	*GET*	*{* *"status": "error", "message": "El registro ya existe en el sistema."* *}* *{* *"status": "ok", "message": "El registro No existe en el sistema."* *}*
Ingresar Formulario	*/API/IngresarFormulario*	*{* *"id": "1234567890",* *"nombres": "Jorge",* *"apellidos": "Villanueva",* *...* *}*	*POST*	*{* *"status": "ok",* *"message": "Registro ingresado correctamente."* *}*
Obtener un Formulario	*/API/ObetenerUnFormulario/{id}*	*{id}*	*GET*	*{* *"id": "1234567890",* *"nombres": "Jorge",* *"apellidos": "Villanueva",* *...* *}*
Obtener Todos los Formularios	*/API/ObtenerTodosLosFormularios/{*}*	*{*}*	*GET*	*[* *{* *"id": "1234567890",* *"nombres": "Jorge",* *"apellidos": "Villanueva",* *...* *},* *{* *"id": "1234567890",* *"nombres": "Danny",* *"apellidos": "Mora",* *...* *},* *...* *]*
Obtener Generos	*/API/ObtenerGeneros/{genero}*	*{genero}*	*GET*	*[* *{* *"id": "1234567890",* *"nombres": "Jorge",* *"apellidos": "Villanueva",* *"genero": "Masculino",* *...* *},* *{* *"id": "1234567890",* *"nombres": "Danny",* *"apellidos": "Mora",* *"genero": "Masculino",* *...* *},* *...* *]*

Docente:	Edwin Boza G.	Periodo:	PEL3	Año:	2023
Materia:	Sistemas Distribuidos				
Actividad:	Tarea 03: Diseño e implementación básica del módulo de almacenamiento				

ESQUEMA/ARQUITECTURA

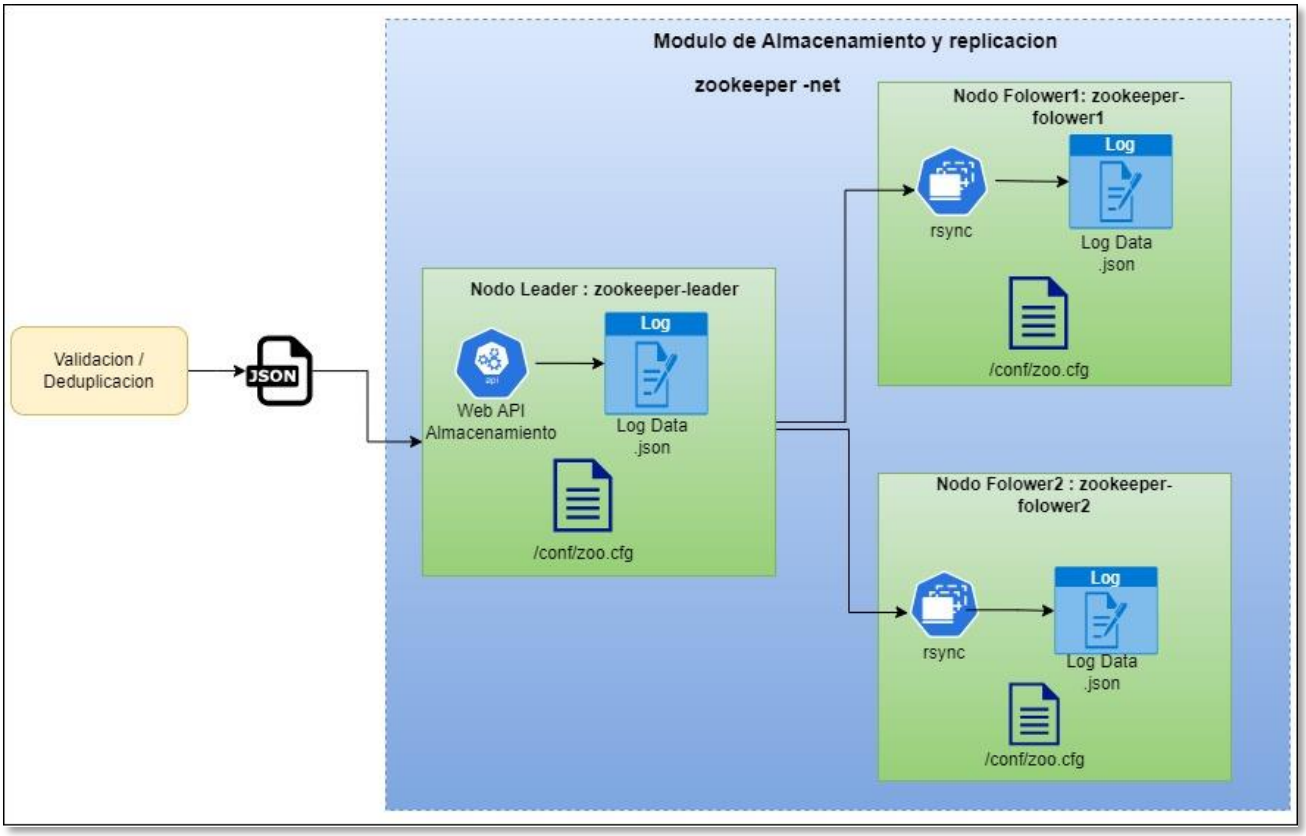


Fig.2 Arquitectura de almacenamiento distribuido

DESCRIPCIÓN DEL MÉTODO DE ALMACENAMIENTO LOCAL:

El módulo almacenamiento que se implementará consiste en tres servidores contenedores: Un contenedor para el nodo líder y dos contenedores para cada uno de los nodos seguidores. Cada instancia del módulo de almacenamiento tendrá su propio almacenamiento local.

El almacenamiento local se implementará utilizando archivos en el sistema de archivos del contenedor. Cada instancia del módulo de almacenamiento tendrá un archivo de registro (log) que se utiliza para almacenar las operaciones realizadas en el almacenamiento. Este archivo de registro es importante para mantener la consistencia de los datos y sincronizar los nodos seguidores con el líder.

El formato del archivo de registro será en texto plano con formato estructurado .JSON. Cada acción se insertará como un registro en el archivo .JSON tales como inserción, actualización o eliminación de datos. Cada registro contiene información como la clave del dato, el valor y el tipo de operación realizada.

Docente:	Edwin Boza G.	Periodo:	PEL3	Año:	2023
Materia:	Sistemas Distribuidos				
Actividad:	Tarea 03: Diseño e implementación básica del módulo de almacenamiento				

DESCRIPCIÓN DEL MÉTODO DE REPLICACIÓN:

Modelo de replicación:

El modelo de replicación que se implementará en el almacenamiento distribuido es el modelo de replicación líder-seguidor con sincronización tipo síncrona.

A continuación, se proporciona una descripción del modelo, incluyendo el método de comunicación entre las instancias y el proceso para realizar la replicación y responder al cliente:

1. ***Sincronización Síncrona:***

El líder es responsable de recibir todas las operaciones de escritura y mantener actualizado el estado de los datos. Cada vez que se realiza una operación de escritura en el líder, esta se propaga de manera síncrona a todos los seguidores. Los seguidores aplican las operaciones en el mismo orden en que se recibieron del líder, lo que garantiza la coherencia de los datos en todas las réplicas.

La sincronización síncrona implica que el líder esperará que los nodos seguidores confirmen la recepción y la escritura exitosa antes de considerar que la operación está completa. Esto asegurará que todas las réplicas mantengan una copia actualizada y consistente de los datos en todo momento.

2. ***Método de Comunicación:***

Coordinación de nodos líder-seguidores:

ZooKeeper es un servicio centralizado y altamente disponible utilizado para la coordinación y gestión de aplicaciones distribuidas. Proporciona un conjunto de servicios de coordinación, como almacenamiento de datos, notificaciones y bloqueo distribuido.

En nuestro proyecto se utilizará Zookeeper como servicio centralizado para la coordinación de la sincronización entre los nodos líder-seguidores

PROCESO DE REPLICACIÓN Y RESPUESTA AL CLIENTE:

A continuación, se detalla de manera general el proceso en que funcionará el modulo de almacenamiento con su respectiva replicación entre nodos

1. *El líder recibe la operación de escritura de la API expuesta y la procesa en su almacenamiento local.*
2. *El líder genera un mensaje que contiene los detalles de la operación de escritura y lo envía a los seguidores correspondientes a través del protocolo de transferencia de archivos rsync que está diseñado para sincronizar archivos de manera rápida y solo enviar las diferencias entre los archivos.*
3. *Los seguidores reciben los datos del líder y sincronizan sus archivos.*
4. *Cada seguidor actualiza su almacenamiento local con la operación recibida.*
5. *Los seguidores indican al líder el estado de la sincronización a través de una verificación basada en el hash de los archivos*
6. *Una vez que los seguidores han aplicado la operación de escritura en su almacenamiento local, el líder responde al cliente indicando que la operación fue exitosa o el resultado de la sincronización con los seguidores.*
7. *El líder confirma la operación en su almacenamiento local después de enviar la respuesta al cliente.*

UNIVERSIDAD DE ESPECIALIDADES ESPÍRITU SANTO

Facultad de Ingeniería

Docente:	Edwin Boza G.	Periodo:	PEL3	Año:	2023
Materia:	Sistemas Distribuidos				
Actividad:	Tarea 03: Diseño e implementación básica del módulo de almacenamiento				

A continuación, se detalla parte del código Python usado para la sincronización de los archivos desde el nodo líder a los nodos seguidores.

- *La biblioteca estándar shutil se utilizará para la sincronización de archivos.*
- *Se definirá la ubicación de los archivos en el nodo líder y los nodos seguidores.*

```
leader_directory = '/ruta/al/nodo/lider'

follower_directories = ['/ruta/al/nodo/seguidor1', '/ruta/al/nodo/seguidor2', '/ruta/al/nodo/seguidor3']
```

- *Se utiliza **shutil.rmtree** para eliminar el contenido de los directorios de los seguidores y **shutil.copytree** para copiar los archivos desde el nodo líder a los nodos seguidores. La función sync_files se ejecuta en un bucle continuo, lo que permite mantener la sincronización en tiempo real entre los nodos*

```
def sync_files():
    for follower_dir in follower_directories:
        try:
            shutil.rmtree(follower_dir)  # Elimina el contenido del directorio del seguidor
            shutil.copytree(leader_directory, follower_dir)  # Copia los archivos desde el líder al seguidor
        except FileNotFoundError:
            os.makedirs(follower_dir)  # Crea el directorio del seguidor si no existe
```

```
while True:
    sync_files()  # Sincroniza los archivos
    time.sleep(5)  # Espera un intervalo de tiempo antes de volver a sincronizar
```

- *Para verificar si la sincronización de datos se realiza satisfactoriamente entre los nodos, se utilizará una verificación basada en el hash de los archivos. A continuación, se muestra el código que se usará entre los nodos para realizar esta verificación.*

  NOTA. Los eventos de sincronización se registran utilizando los métodos error() e info() del objeto de registro. Los mensajes de error se guardan en el archivo de registro cuando ocurre una falla en la sincronización, mientras que los mensajes de información se guardan para indicar que la sincronización ha sido satisfactoria.

Docente:	Edwin Boza G.	Periodo:	PEL3	Año:	2023
Materia:	Sistemas Distribuidos				
Actividad:	Tarea 03: Diseño e implementación básica del módulo de almacenamiento				

```
import hashlib
import logging


def calculate_file_hash(file_path):
    """
    Calcula el hash de un archivo dado.
    """
    hasher = hashlib.md5()
    with open(file_path, 'rb') as file:
        for chunk in iter(lambda: file.read(4096), b''):
            hasher.update(chunk)
    return hasher.hexdigest()


def check_data_sync():
    """
    Verifica la sincronización de datos entre los nodos líder y seguidores.
    """
    leader_files = os.listdir(leader_directory)
    logger = logging.getLogger('sync_logger')
    logger.setLevel(logging.INFO)
    file_handler = logging.FileHandler('sync_log.txt')
    logger.addHandler(file_handler)


    for follower_dir in follower_directories:
        follower_files = os.listdir(follower_dir)
        for file in leader_files:
            leader_file_path = os.path.join(leader_directory, file)
            follower_file_path = os.path.join(follower_dir, file)
            if file in follower_files:
```

Docente:	Edwin Boza G.	Periodo:	PEL3	Año:	2023
Materia:	Sistemas Distribuidos				
Actividad:	Tarea 03: Diseño e implementación básica del módulo de almacenamiento				

```
            leader_hash = calculate_file_hash(leader_file_path)
            follower_hash = calculate_file_hash(follower_file_path)
            if leader_hash != follower_hash:
                logger.error(f"La sincronización ha fallado para el archivo {file}")
            else:
                logger.info(f"La sincronización ha sido satisfactoria para el archivo {file}")
        else:
            logger.error(f"El archivo {file} no se ha replicado en el nodo seguidor")


# Configuración del logger
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')


# Ejemplo de uso
check_data_sync()
```

DESCRIPCIÓN DE LAS LIMITACIONES DE LA REPLICACIÓN

En el caso del modelo de replicación líder-seguidor con sincronización síncrona, existen ciertas limitaciones a considerar:

1. ***Capacidad limitada de escritura:*** *En este modelo de replicación, sólo el nodo líder está autorizado para realizar operaciones de escritura, mientras que los nodos seguidores se utilizan para leer y replicar los datos. Esto limita la capacidad de escritura del sistema a la capacidad del nodo líder. Si el nodo líder experimenta una alta carga de escritura o se convierte en un cuello de botella, puede generar un punto único de falla y afectar el rendimiento general del sistema.*

   Para tratar de contener este punto se implementará un esquema de transferencia incremental eficiente a través de la función en Python "rsync".

2. ***Consistencia eventual:*** *En un entorno de replicación distribuida, la consistencia eventual es un enfoque comúnmente utilizado para mantener la coherencia de los datos entre los nodos. Esto significa que después de realizar una operación de escritura, los nodos seguidores pueden tardar un tiempo en recibir y replicar los cambios. Durante este período, es posible que los nodos seguidores proporcionen datos desactualizados a los clientes, lo que puede resultar en inconsistencias temporales.*

3. ***Escalabilidad limitada:*** *A medida que aumenta el número de nodos seguidores en el sistema de replicación, la latencia y el ancho de banda requeridos para mantener la coherencia entre ellos pueden convertirse en un desafío. Además, si el nodo líder falla, se requiere un proceso de*

Docente:	Edwin Boza G.	Periodo:	PEL3	Año:	2023
Materia:	Sistemas Distribuidos				
Actividad:	Tarea 03: Diseño e implementación básica del módulo de almacenamiento				

elección de un nuevo líder, lo que puede introducir cierto tiempo de inactividad y afectar la disponibilidad del sistema.

4. ***Dependencia de ZooKeeper:*** *La coordinación entre los nodos líder y seguidores se logra mediante el uso de ZooKeeper, que actúa como un servicio centralizado para el registro, la sincronización y la elección del líder. Si ZooKeeper experimenta problemas de rendimiento o fallas, puede afectar directamente la operación del sistema de replicación. Además, el rendimiento y la escalabilidad de ZooKeeper también pueden ser un factor limitante en el sistema global.*

DESRIPCIÓN DE LA COORDINACIÓN DE LAS RÉPLICAS.

En el modelo de replicación líder-seguidor descrito, la coordinación de las réplicas se lleva a cabo utilizando el servicio de ZooKeeper. A continuación, se detalla el proceso de coordinación:

Identificación del líder:

ZooKeeper se encarga de la elección del líder, al inicio todos los nodos (líder y seguidores) se registran en ZooKeeper. este utiliza algoritmos de elección de líder, como el algoritmo de elección de coordinador, para seleccionar aleatoriamente a uno de los nodos como líder, el nodo seleccionado aleatoriamente como líder es notificado y asume el rol de líder en el sistema.

Identificación de los seguidores:

Una vez que el líder ha sido seleccionado, los seguidores también se registran en ZooKeeper, los seguidores establecen una conexión con ZooKeeper para mantenerse actualizados sobre el líder actual y recibir notificaciones de cambios en la estructura del sistema.

Conocimiento del líder por parte de los seguidores:

Los seguidores consultan periódicamente ZooKeeper para obtener la información del líder actual, ZooKeeper proporciona los detalles del líder a los seguidores, como su dirección IP y puerto, para que puedan establecer comunicación con él.

ZooKeeper actúa como un componente externo para la coordinación del sistema distribuido, Proporciona servicios de descubrimiento y coordinación, como la elección del líder y la detección de réplicas caídas, los nodos líder y seguidores mantienen una conexión activa con ZooKeeper para recibir notificaciones y actualizar su estado en caso de cambios en la estructura del sistema.

Detección de una réplica caída:

ZooKeeper supervisa el estado de los nodos y detecta si un líder o seguidor deja de responder o se desconecta, en caso de que el líder falle, ZooKeeper realiza una nueva elección de líder y notifica al nuevo líder y a los seguidores, los seguidores reciben la notificación de cambio de líder y actualizan su conocimiento sobre el nuevo líder para establecer comunicación con él.

Actualización de una réplica:

Cuando se realiza una operación de escritura en el líder, este envía mensajes a las colas correspondientes a los seguidores para informarles sobre las actualizaciones realizadas, los seguidores reciben los mensajes y actualizan su almacenamiento local para mantenerse sincronizados con el líder.

Promoción de un seguidor a líder:

En caso de que el líder falle o se desconecte, ZooKeeper realiza una nueva elección de líder entre los seguidores restantes, selecciona aleatoriamente a uno de los seguidores como el nuevo líder, el seguidor seleccionado como nuevo líder es notificado y asume el rol de líder en el sistema.

UNIVERSIDAD DE ESPECIALIDADES ESPÍRITU SANTO

Facultad de Ingeniería

Docente:	**Edwin Boza G.**	**Periodo:**	**PEL3**	**Año:**	**2023**
Materia:	**Sistemas Distribuidos**				
Actividad:	**Tarea 03: Diseño e implementación básica del módulo de almacenamiento**				

REPORTE DEL AVANCE DEL PROYECTO

URL Google Drive

https://docs.google.com/document/d/1ewV6wMqFOZcnxWMq4R2BNF3N66s-hHQBY3Gjmc79qRU/edit#heading=h.9aqsy6fnsoeg

URL Github

https://github.com/diem1285/TAREA-2-Comunicacion_indirecta

AVANCE GENERAL DEL PROYECTO.

1. *Implementación del sistema operativo Ubuntu*
2. *Implementación del módulo de capturador de datos*
3. *Instalación de sincronía asíncrona mediante el uso de colas de mensajes haciendo uso del software para gestión de mensajes RabbitMQ.*
4. *Instalación de contenedores DOCKER.*
5. *Implementación de módulo para duplicación y validación de datos.*
6. *Se elige la arquitectura de que método de replicación usará el modelo de almacenamiento distribuido*
7. *Reuniones de diseño para el proyecto de implementación de sistema de almacenamiento distribuido*
8. *Se empieza armar el código de replicación de datos entre los nodos líder -seguidores en lenguaje Python*
9. *Instalación de Apache ZooKeeper.*
10. *Se realiza documentación del proyecto.*

Pendientes:

- *Implementación de concurrencia (que se puedan ejecutar varias instancias en forma simultánea)*
- *Implementación de API Rest, no se ha implementado pues se está esperando que se apruebe la arquitectura planteada en el presente modelo.*