Arquitectura MVC y Bases de Datos Distribuidas Avanzadas

Objetivo:

Desarrollar una aplicación distribuida que utilice la arquitectura MVC para gestionar la información de un sistema hospitalario integrado por múltiples centros médicos. La solución debe exponer servicios mediante API RESTful, garantizar la replicación y sincronización de datos en bases de datos distribuidas y enfrentar desafíos avanzados como la selección del modelo de distribución, configuración en contenedores y esquemas de replicación.

Parte 1: Configuración de la Infraestructura

---

1. **Máquinas Virtuales en la Nube:**
    - Crear al menos tres máquinas virtuales (VM) que representen centros médicos.
    - Designar una VM principal (ejemplo: "Centro Médico Quito") que contenga la base de datos central y otras VMs (ej.: "Centro Médico Guayaquil" y "Centro Médico Cuenca") para almacenar bases de datos locales donde se registrarán las consultas.
    - Configurar redes virtuales (VNet) y aplicar grupos de seguridad (NSG) para garantizar una comunicación segura entre nodos.
2. **Bases de Datos Distribuidas y Aspectos Avanzados:**
    - **Instalación y Configuración:** Instalar y configurar un SGBD como MariaDB (o PostgreSQL/MySQL) en cada VM para permitir la replicación distribuida.
    - **Replicación y Distribución de Datos:**
        - **Modelo Centralizado vs. Distribuido:**
        - Explorar las ventajas y desafíos de un modelo centralizado (donde la mayor parte de la información se gestiona en un único nodo) versus un modelo distribuido, en el que los datos se reparten y replican en nodos independientes para mejorar la disponibilidad y escalabilidad.
        - **Tipos de Bases de Datos:**
            - *Homogéneas:* Todas las bases de datos emplean el mismo SGBD, lo que facilita la replicación y consistencia.

- *Heterogéneas:* Uso de diferentes SGBD en distintos nodos, lo que puede requerir herramientas de interoperabilidad y adaptación en la comunicación.

- **Configuración de un Nodo Distribuido en Docker:** Como alternativa o complemento a la configuración en VM, se deberá configurar uno o más nodos utilizando contenedores Docker. Esto permitirá el despliegue rápido, el escalado y la replicación de los servicios en entornos aislados y reproducibles.

- **Métodos de Distribución:**

  - *Fragmentación Horizontal:* Dividir las filas de una tabla entre distintos nodos, permitiendo que cada nodo almacene un subconjunto de registros.

  - *Fragmentación Vertical:* Distribuir las columnas de una tabla entre nodos distintos, optimizando el almacenamiento dependiendo del uso de los datos.

  - *Fragmentación Híbrida:* Combinación de los dos métodos anteriores, aplicando técnicas según las necesidades específicas de la consulta y el almacenamiento.

- **Esquemas de Replicación:**

  - *Activa/Asistida vs. Pasiva:*

    - En la replicación **activa**, todos los nodos pueden aceptar operaciones de escritura y lectura, mientras que en la **pasiva** se designa un nodo maestro donde se realizan las escrituras y los esclavos únicamente replican la información.

  - *Sincrónica vs. Asincrónica:*

    - La replicación **sincrónica** garantiza que todos los nodos se actualicen en tiempo real con cada transacción, mientras que la **asincrónica** puede presentar ligeros desfases temporales entre nodos, aunque generalmente mejora el rendimiento en escenarios de alta carga.

- Configurar el acceso remoto a las bases de datos para facilitar la gestión distribuida y el acceso centralizado a la información global (empleados, médicos, especialidades).

Parte 2: Desarrollo del Backend con Arquitectura MVC

1. **Diseño de la Aplicación Monolítica MVC:**
   - Utilizar Node.js con Express para desarrollar una aplicación monolítica que implemente el patrón Modelo-Vista-Controlador:
     - **Modelo (Model):** Encargado de la interacción con las bases de datos distribuidas (tanto la base central como las locales), incluyendo la lógica necesaria para manejar la replicación y la distribución de datos.
     - **Vista (View):** Aunque la aplicación exponga principalmente respuestas en formato JSON (por ser una API RESTful), se debe estructurar la lógica de presentación para dar coherencia en las respuestas y facilitar la integración.
     - **Controlador (Controller):** Define los endpoints de la API RESTful para las operaciones CRUD sobre las entidades:
       - Centros Médicos
       - Médicos
       - Especialidades
       - Empleados
       - Consultas Médicas
2. **API RESTful y Documentación:**
   - Desarrollar endpoints que permitan crear, leer, actualizar y eliminar registros.
   - Asegurar la integración entre los modelos y las bases de datos, coordinando la replicación y las consultas de datos globales y locales.
   - Documentar la API utilizando herramientas como Swagger, detallando cada endpoint, parámetros y respuestas esperadas.

Parte 3: Desarrollo del Frontend Interfaz de Usuario para Administración:

- Desarrollar una interfaz web (por ejemplo, en Vue.js) que permita gestionar centros médicos, empleados, médicos y especialidades, con formularios de creación, edición y eliminación, y vistas para reportes y estadísticas.

2. **Interfaz de Usuario para Hospitales:**
   - Crear una aplicación web (por ejemplo, en React) para la gestión de consultas médicas, garantizando la autenticación para que cada usuario acceda solo a la información correspondiente a su centro.
3. **Integración con la API:**
   - Los frontends deberán consumir de manera segura y eficiente los endpoints definidos en la API RESTful del backend.

Parte 4: Despliegue y Pruebas

---

1. **Despliegue en la Nube (Azure) y Uso de Contenedores:**
   - Gestionar todo el proyecto a través de Azure DevOps, que incluirá el control de versiones, pipelines de integración y despliegue continuo (CI/CD).
   - Desplegar las máquinas virtuales y/o contenedores Docker (para nodos distribuidos) en Azure.
   - Configurar servicios adicionales como Azure Load Balancer para asegurar la disponibilidad y escalabilidad de la aplicación.
2. **Pruebas de Integración y Funcionales:**
   - Realizar pruebas que verifiquen la correcta replicación entre bases de datos (maestro y esclavos) y que validen que la fragmentación y distribución de datos se han implementado de forma eficaz.
   - Probar todos los endpoints de la API RESTful para confirmar que las operaciones CRUD se ejecutan conforme a lo esperado.
   - Validar la funcionalidad y usabilidad de las interfaces web.
3. **Documentación Técnica Final:**
   - Entregar una documentación completa que detalle:
     - La configuración de la infraestructura, bases de datos y replicación.

- Las decisiones en torno al modelo centralizado vs. distribuido.
- La implementación de fragmentación horizontal, vertical e híbrida.
- El esquema de replicación seleccionado (activa/pasiva y sincrónica/asincrónica).
- La configuración y despliegue de nodos distribuidos en Docker.
- Diagramas de arquitectura y flujos de datos.
- La documentación de la API (Swagger).

Criterios de Evaluación:

- **Infraestructura y Configuración de Bases de Datos:**
  - Correcta instalación y configuración de las VMs y/o contenedores Docker.
  - Implementación adecuada de la replicación distribuida, teniendo en cuenta el modelo centralizado vs. distribuido.
  - Aplicación de métodos de fragmentación (horizontal, vertical, híbrida) y selección apropiada del esquema de replicación (activa/pasiva, sincrónica/asincrónica).
- **Diseño y Desarrollo del Backend (MVC):**
  - Implementación correcta del patrón MVC en Node.js/Express.
  - Endpoints RESTful completos y funcionales, con integración adecuada con las bases de datos distribuidas.
- **Desarrollo del Frontend:**
  - Interfaces intuitivas y completas para la administración y la gestión hospitalaria, que consuman de forma segura la API.
- **Integración y Pruebas:**
  - Validación de la comunicación entre los componentes del sistema y pruebas de replicación distribuidas.
  - Confirmación de la correcta operación de la aplicación en diferentes entornos (local y en la nube).
- **Documentación y Uso de Azure DevOps:**
  - Entrega de una documentación técnica detallada, diagramas de arquitectura y flujos de datos.

- Gestión del proyecto en Azure DevOps, incluyendo control de versiones y pipelines CI/CD.

Instrucciones Adicionales:

- El proyecto debe ser desarrollado en equipos utilizando herramientas de control de versiones (por ejemplo, Git) y gestionado a través de Azure DevOps.
- Se recomienda elaborar diagramas que ilustren tanto la arquitectura MVC de la aplicación como la distribución y replicación de bases de datos (incluyendo modelos de fragmentación y esquemas replicados).
- La documentación técnica debe incluir detalles sobre la configuración del entorno Docker para la implementación de nodos distribuidos, y explicar las ventajas y limitaciones de cada modelo y método utilizado.

Esquema de Evaluación

El proyecto se divide en tres grandes bloques evaluativos, cada uno de ellos con criterios específicos. Cada criterio se calificará de 0 a 2, siendo:

- 2: Excelente, cumple todos los requisitos y demuestra un alto grado de calidad.
- 1: Aceptable, cumple parcialmente los requisitos o requiere mejoras.
- 0: Insuficiente, no cumple lo solicitado o presenta fallas críticas.

Los tres bloques evaluativos son:

1. **Infraestructura y Configuración de Bases de Datos Distribuidas**
   Aspectos a evaluar:
   - **Instalación y configuración de VMs y/o contenedores Docker:**
     - Se evaluará la correcta creación de nodos (máquinas virtuales y/o contenedores) y su adecuada integración en la red (VNet, NSG).
   - **Modelos de distribución de la BDD:**
     - Implementación y justificación del modelo centralizado vs. distribuido.
     - Uso de fragmentación horizontal, vertical o híbrida según el caso.
   - **Esquemas de replicación:**
     - Configuración y documentación de la replicación (activa/pasiva y sincrónica/asincrónica) en las bases de datos homogeneas u heterogéneas.

2. **Desarrollo del Backend y Frontend (Arquitectura MVC y API RESTful)**
   Aspectos a evaluar:
   - **Implementación de la arquitectura MVC en el backend:**
     - Desarrollo de modelos, vistas (respuestas JSON) y controladores que permitan operaciones CRUD en las entidades (centros médicos, médicos, especialidades, empleados, consultas médicas).
   - **Implementación de la API RESTful:**
     - Definición clara y funcional de los endpoints, uso correcto de métodos HTTP y documentación (por ejemplo, mediante Swagger).
   - **Desarrollo e integración del Frontend:**
     - Interfaces web (por ejemplo, con Vue.js para administración y React para los hospitales) que consuman de manera segura la API y proporcionen una experiencia de usuario adecuada.
3. **Documentación Técnica y Gestión del Proyecto (Azure DevOps y Entorno CI/CD)**
   Aspectos a evaluar:
   - **Documentación de la solución:**
     - Elaboración de diagramas de arquitectura que muestren el flujo de datos, la configuración de replicación y la distribución de la información.
     - Descripción detallada de los modelos utilizados (centralizado vs. distribuido, fragmentación, replicación, configuración Docker).
   - **Gestión del proyecto en Azure DevOps:**
     - Uso adecuado de control de versiones, integración y despliegue continuo (CI/CD) y compartición de documentación en el repositorio.

Parámetros de Calificación (Cada criterio se evalúa de 0 a 2):

Bloque	Criterio	Puntuación (0-2)
1. Infraestructura y Configuración de BDD	Instalación y configuración de VMs y/o Docker	0 = Inexistente / 1 = Parcial / 2 = Completo y correcto

Bloque	Criterio	Puntuación (0-2)
	Implementación del modelo centralizado vs. distribuido y fragmentación	0 = No aplicable / 1 = Parcial / 2 = Bien fundamentado y aplicado
	Configuración de esquemas de replicación activa/pasiva y sincrónica/asincrónica	0 = Incorrecto / 1 = Incompleto / 2 = Cumple todos los parámetros
2. Desarrollo del Backend y Frontend	Implementación de la arquitectura MVC y API RESTful	0 = Fallida / 1 = Parcial / 2 = Excelente y funcional
	Desarrollo y calidad de interfaces web (Frontend)	0 = Deficiente / 1 = Aceptable / 2 = Intuitivas y bien integradas
3. Documentación y Gestión del Proyecto	Calidad y completitud de la documentación técnica (diagramas, explicaciones)	0 = Inexistente / 1 = Limitada / 2 = Completa y clara
	Uso de Azure DevOps (control de versiones, pipelines CI/CD)	0 = No se utiliza / 1 = Uso parcial / 2 = Manejo adecuado y exhaustivo

Cada bloque podrá tener un peso específico en la nota final. Por ejemplo, si se decide que cada bloque representa un tercio de la calificación global, se sumarán las puntuaciones y se promediarán (si se suman 2 criterios por bloque, la máxima del bloque será 4 puntos, etc.).