**Sección de Metodología en la Arquitectura**

**Metodología de Desarrollo**  
El proyecto seguirá una combinación de **Agile** y **DevOps**:

* **Agile (Scrum)**:
  + El trabajo se organizará en **sprints de 1 semanas**.
  + Cada microservicio será gestionado por un equipo independiente.
  + **Reuniones semanales o puntuales** se realizarán con los equipos responsables de microservicios relacionados, como **usuarios, notificaciones, mensajería y reportes**, para asegurar que la integración y comunicación entre servicios sea fluida.

**Descripción de publicaciones y comentarios**

El microservicio de Publicaciones y Comentarios en una red social maneja la creación, gestión y visualización de publicaciones y sus comentarios, permitiendo interacciones como reacciones (likes) y respuestas en cada comentario.

**1. Descomposición del Microservicio**

**a) Publicaciones**

* Crear, actualizar, eliminar y obtener publicaciones.
* Almacenar metadatos (usuario, fecha, etiquetas, imágenes/videos adjuntos).
* Soporte para interacciones: "me gusta", compartir.

**b) Comentarios**

* Asociados a publicaciones (o respuestas anidadas entre comentarios).
* CRUD (Create, Read, Update, Delete) de comentarios.
* Gestión de likes o reacciones.

**1.2. Comunicación con Otros Microservicios**

El microservicio debe interactuar con:

1. **Microservicio de Usuarios**:
   * Verifica la identidad y obtiene datos del autor (nombre, avatar, etc.) mediante **API REST**.
2. **Microservicio de Notificaciones y Mensajería**:
   * Envía notificaciones cuando se agregan comentarios o likes mediante **Kafka/RabbitMQ**.
3. **Microservicio de Reportes**:
   * Publicaciones o comentarios pueden ser reportados; el evento es enviado al servicio de reportes.

**RabbitMQ**

**¿Qué es?**

RabbitMQ es un **sistema de mensajería** basado en colas, que permite que diferentes partes de tu aplicación (o microservicios) se comuniquen de forma **asíncrona**. Esto significa que un servicio puede enviar mensajes sin esperar una respuesta inmediata, lo que mejora la eficiencia.

**¿Por qué usar RabbitMQ?**

* **Desacoplamiento**: Los microservicios pueden funcionar independientemente sin necesidad de conocer el estado del otro.
* **Gestión de Mensajes**: RabbitMQ garantiza que los mensajes lleguen incluso si un servicio está inactivo temporalmente.
* **Escalabilidad**: Soporta millones de mensajes por segundo.

**Uso en tu proyecto:**

* **Publicaciones y comentarios** envía mensajes a **notificaciones y mensajería** cada vez que alguien comenta o reacciona a una publicación.
* **Reportes** recibe eventos si una publicación es denunciada.

**Flujo de Mensajes con RabbitMQ:**

1. El microservicio de **publicaciones** envía un mensaje al intercambiador (exchange).
2. El intercambiador redirige el mensaje a una **cola** específica.
3. El microservicio de **notificaciones** consume los mensajes de la cola para enviar las alertas.

**2. Diseño de Base de Datos (MongoDB)**

1. El microservicio usará **MongoDB Compass** para gestionar las siguientes colecciones:

**Colección**: posts

{

"\_id": "postId123",

"authorId": "userId123",

"content": "Texto de la publicación",

"media": ["img1.jpg", "video1.mp4"],

"tags": ["viajes", "aventura"],

"createdAt": "2024-10-28T10:00:00Z",

"likes": ["userId456", "userId789"],

"commentCount": 5,

"reports": ["reportId1", "reportId2"]

}

**Colección**: comments

{

"\_id": "commentId1",

"postId": "postId123",

"authorId": "userId789",

"content": "¡Gran publicación!",

"createdAt": "2024-10-28T10:10:00Z",

"likes": ["userId123", "userId456"],

"reports": ["reportId3"]

}

**3. API REST: Endpoints Definidos**

**Publicaciones**

* **POST /posts**: Crear una nueva publicación.
* **GET /posts**: Obtener todas las publicaciones (con paginación).
* **GET /posts/{id}**: Consultar una publicación específica.
* **PUT /posts/{id}**: Actualizar una publicación.
* **DELETE /posts/{id}**: Eliminar una publicación.

**Comentarios**

* **POST /posts/{postId}/comments**: Agregar un comentario.
* **GET /posts/{postId}/comments**: Listar comentarios de una publicación.
* **DELETE /comments/{commentId}**: Eliminar un comentario.

**4. Comunicación Asíncrona (Eventos con Kafka/RabbitMQ)**

* **Nuevo Comentario**: Envía un evento al servicio de notificaciones para avisar al autor.
* **Like o Reacción**: Genera un evento para notificaciones.
* **Denuncia de Contenido**: Notifica al servicio de reportes cuando se reporta una publicación o comentario.

**5. Seguridad y Autenticación**

* **JWT Tokens**: Cada solicitud incluye un token que identifica al usuario.
* **Autorización**:
  + Solo el autor puede eliminar su publicación/comentario.
  + Se consulta al microservicio de **usuarios** para verificar el rol del usuario (ej.: administrador).

**6. Escalabilidad y Despliegue**

* **Contenedores Docker**: Cada instancia del microservicio es independiente.
* **Orquestación con Kubernetes**: Gestiona múltiples instancias para soportar alta carga.
* **Redis**: Cachea publicaciones populares para mejorar la velocidad.

Con esta arquitectura, tu microservicio de **publicaciones y comentarios** estará organizado en módulos claros, integrará **comunicación asíncrona** con otros servicios y será escalable con **Docker y Kubernetes**.