Project Documentation: Comment Microservice for Jira Clone

Overview

This project is a microservices-based application designed for handling comments, tasks, and user management in a Jira clone. The application utilizes various technologies to provide a scalable, secure, and efficient service. Kafka is used for inter-service communication, and Docker is employed for containerization and deployment.

Kafka Setup: Configures Kafka producer and consumer.
Topic Management: Handles creation and configuration of Kafka topics.
Consumer Logic: Listens to Kafka topics and processes messages.
Error Handling: Logs errors and warnings for Kafka operations.

Tech Stack

- **Node.js**: JavaScript runtime for building scalable network applications.
- **Express**: Web framework for building RESTful APIs.
- **Kafka**: Distributed event streaming platform for handling real-time data feeds.
- **Nginx**: Web server and reverse proxy for load balancing and HTTP routing.
- **JWT (JSON Web Token)**: Authentication mechanism for secure API access.
- **Docker**: Containerization platform for consistent development and deployment.
- **Winston**: Logging library for Node.js to manage application logs.
- **Morgan**: HTTP request logger middleware for Node.js.
- **Mongoose**: ODM (Object Data Modeling) library for MongoDB and Node.js.
- **Other Libraries**: Various additional libraries as needed

Project Structure.

```
CommentService
├── configs
├── logs
├── messageQueue
├── migrations
├── models
├── node_modules
├── route
├── services
├── Dockerfile
├── index.js
├── migrate-mongo-config.js
├── package.json
├── package-lock.json
├── README.md
└── test.js

KafkaService
├── Admin.js
├── Dockerfile
├── index.js
├── logging.service.js
├── node_modules
├── package.json
├── package-lock.json
└── utilities

logs
└── (log files)

nginx-docker
├── Dockerfile
├── index.html
├── nginx.conf

README.Docker.md
README.md

scripts
└── run_Kafka.sh
```

```
TaskService
├── configs
├── logs
├── migrations
├── models
├── node_modules
├── route
├── services
├── Dockerfile
├── index.js
├── migrate-mongo-config.js
├── package.json
├── package-lock.json
├── README.md
└── test.js

UserService
├── configs
├── migrations
├── models
├── node_modules
├── route
├── services
├── Dockerfile
├── index.js
├── migrate-mongo-config.js
├── package.json
├── package-lock.json
└── README.md

compose.yaml
index.html
```

Setup and Installation

1. **Clone the Repository**

```
git clone <repository-url>
cd <repository-directory>
```

2. **Install Dependencies**

```
cd <microservice>
npm install
```

3. **Configure Environment Variables**
    - **KAFKA ADMIN SERVICE (KAFKA Service for Topic Creation)**

      create `.env.local` file in the **KafkaService** folder with the following content:

```
# Kafka Configuration
KAFKA_BROKER=kafka:9092 # Replace with your actual broker address

KAFKA_CLIENT_ID=kafka-service

# Comment Microservice Topics
PORT= 9000
```

    - **COMMENT SERVICE**

      create `.env.local` file in the project **CommentService** folder with the following content:

```
## Kafka Configuration

# Kafka Broker
KAFKA_BROKER=kafka:9092
# Replace with your actual broker address
KAFKA_CLIENT_ID=comment-service
KAFKA_GROUP_ID=comment-group
```

```
# Comment Microservice Topics
COMMENT_ADDED_TOPIC=comment.added
COMMENT_UPDATED_TOPIC=comment.updated
COMMENT_DELETED_TOPIC=comment.deleted

# Task Microservice Topics
TASK_CREATED_TOPIC=task.created
TASK_UPDATED_TOPIC=task.updated
TASK_DELETED_TOPIC=task.deleted

# Service Port
PORT=8003
```

4. **Clone the Repository**

Without Docker:

Starting Commands for Each Service

To start each microservice without Docker, you need to run them individually using Node.js commands. Below are the commands for starting each service:

CommentService

```
cd TaskService
npm install
npm start
```

TaskService

```
cd TaskService
npm install
npm start
```

UserService

```
cd UserService
npm install
npm start
```

1.

KafkaService

```
cd KafkaService
npm install
npm start
```

2.

Steps to Start Kafka and Nginx Without Docker

Kafka Setup

1. **Download and Extract Kafka and Zookeeper**
    - Download Kafka from the Apache Kafka website.
    - Extract the downloaded files to a desired location on your machine.

Start Zookeeper
Navigate to the Kafka directory and run Zookeeper:

```
cd <kafka-directory>
bin/zookeeper-server-start.sh config/zookeeper.properties
```

2.
    - This command starts Zookeeper using the default configuration provided in `config/zookeeper.properties`.

Start Kafka Broker
In a new terminal window, navigate to the Kafka directory and run the Kafka server:

```
cd <kafka-directory>
bin/kafka-server-start.sh config/server.properties
```

3.
    - This starts the Kafka broker using the default configuration provided in `config/server.properties`.

Nginx Setup

1. **Install Nginx**
   Install Nginx using your operating system's package manager:

For Ubuntu/Debian:

```
sudo apt update
sudo apt install nginx
```

For CentOS/RHEL:

```
sudo yum install epel-release
sudo yum install nginx
```

For macOS (using Homebrew):

```
brew install nginx
```

2. **Configure Nginx**

Navigate to the Nginx configuration directory:

```
cd /etc/nginx
```

Open the Nginx configuration file (`nginx.conf` or a custom configuration file) in a text editor:

```
sudo nano /etc/nginx/nginx.conf
```

Update the configuration to route requests to the microservices. Here is an example configuration for proxying requests to your services:
nginx
Copy code

```
events {
}

http {
    upstream auth_service {
        server localhost:8000;
    }

    upstream user_service {
        server localhost:8001;
    }
    upstream task_service {
        server localhost:8002;
    }

    upstream comment_service {
        server localhost:8003;
    }


    server {
        listen 80;
        #Change this to your domain
        server_name localhost;

        location / {
            root /usr/share/nginx/html;
            index index.html;
        }
        location /auth {
            proxy_pass http://auth_service;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
        location /users {
            proxy_pass http://user_service;
            proxy_set_header Host $host;
```

```
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
        location /comments {
            proxy_pass http://comment_service;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
        location /tasks {
            proxy_pass http://task_service;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }

    }
}
```

-

Start Nginx
After configuring Nginx, start it using the following command:

```
sudo service nginx start
```

3.

To restart Nginx after making changes to the configuration:

```
sudo service nginx restart
```

-

4. **Verify Nginx is Running**
    - Check if Nginx is running by accessing your server's IP address or `localhost` in a web browser. You should see the default Nginx welcome page if it's set up correctly.

With Docker:

```
docker-compose up --build
```

API Endpoints

Comment Service

1. **Create Comment**
    - **Endpoint**: `POST /api/comments`
    - **Description**: Creates a new comment associated with a specific task.
2. **Update Comment**
    - **Endpoint**: `PUT /api/comments/:id`
    - **Description**: Updates the text of an existing comment.
3. **Delete Comment**
    - **Endpoint:** `DELETE /api/comments/:id`
    - **Description**: Deletes a specific comment.

Task Service

1. **Create Task**
    - **Endpoint**: `POST /api/tasks`
    - **Description**: Creates a new task with a title and description.
2. **Update Task**
    - **Endpoint**: `PUT /api/tasks/:id`
    - **Description**: Updates the title and description of an existing task.
3. **Delete Task**
    - **Endpoint**: `DELETE /api/tasks/:id`
    - **Description**: Deletes a specific task.

User Service

1. **Create User**
    - **Endpoint**: `POST /api/users`
    - **Description**: Creates a new user with a specified username.

Nginx Configuration for API Services

The Nginx configuration provided routes traffic to the appropriate services based on the URL path. Here's a breakdown:

- `/auth`: For authentication services, proxied to `auth_service` running on `localhost:8000`.
- `/users`: For user-related services, proxied to `user_service` running on `localhost:8001`.
- `/comments`: For comment-related services, proxied to `comment_service` running on `localhost:8003`.
- `/tasks`: For task-related services, proxied to `task_service` running on `localhost:8002`.

Process Flow

---

1. Task Creation Process Flow

Process Overview

- **Frontend Interaction:** Users initiate a task creation request through the frontend application.
- **NGINX Routing:** NGINX directs the request to the Task Service based on the specified URL path.
- **Task Service Processing:**
    - The Task Service validates the incoming request.
    - It publishes a TASK_CREATION message to the Kafka topic.
    - The service responds to the frontend with a status of 202 Accepted, indicating asynchronous processing.
- **Kafka Message Handling:**
    - The Task Service publishes taskCreated events to the Kafka topic.
- **Task Processing and Database Operations:**
    - Multiple instances of TaskProcessor (or TaskService instances) consume messages from the TASK_CREATION topic.
    - Tasks are created asynchronously in MongoDB.
- **Completion Notification (Optional):**
    - The TaskService may publish an event to notify other services once the task creation is completed.

---

2. Comment Creation Process Flow

Process Overview

- **Frontend Interaction:** Users submit a comment for a specific task.
- **NGINX Routing:** NGINX routes the request to the Comment Service.
- **Comment Service Processing:**
    - The Comment Service validates the comment request.
    - It publishes a COMMENT_CREATION message to Kafka.
- **Kafka Message Handling:**
    - The Comment Service publishes commentCreated events to the Kafka topic.
- **Comment Processing and Database Operations:**
    - Comment Service instances consume messages from the COMMENT_CREATION topic.
    - Comments are saved in MongoDB.
- **Completion Notification (Optional):**
    - The CommentService can notify other services once the comment is successfully created.

---

3. Attachment Upload Process Flow

Process Overview

- **Frontend Interaction:** Users upload attachments related to tasks or comments.
- **NGINX Routing:** NGINX manages the request routing to the Attachment Service.
- **Attachment Service Processing:**
    - The service handles the file upload process.
    - Metadata (e.g., file name and path) is saved and a message (ATTACHMENT_UPLOAD) is published to Kafka.
- **Kafka Message Handling:**
    - The Attachment Service publishes attachmentUploaded events to the Kafka topic.
- **Attachment Processing and Database Operations:**

        - Attachment Service instances consume messages from the ATTACHMENT_UPLOAD topic.
        - Metadata is stored in MongoDB.
- **Completion Notification (Optional):**
    - The AttachmentService can notify users or other services upon upload completion.

---

4. User Management Process Flow

Process Overview

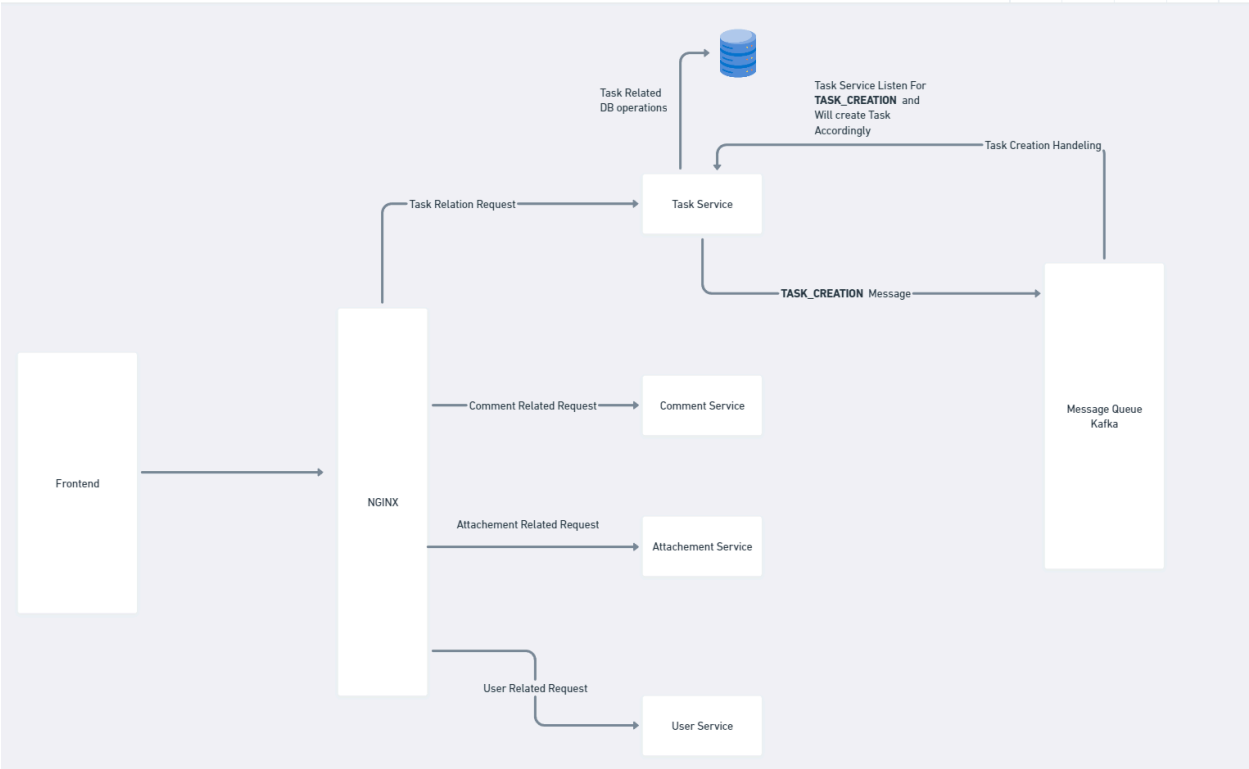
- **Frontend Interaction:** Admins or users submit requests to create or update user details.
- **NGINX Routing:** NGINX directs the request to the User Service.
- **User Service Processing:**
    - The User Service processes user creation/updating, validating the request as necessary.
    - It publishes either a USER_UPDATE or USER_CREATION message to Kafka.
- **Kafka Message Handling:**
    - The User Service publishes userUpdated or userCreated events to the respective Kafka topic.
- **User Processing and Database Operations:**
    - User Service instances consume messages from the related Kafka topic.
    - User details are updated or created in MongoDB.
- **Completion Notification (Optional):**
    - The UserService may notify other services regarding changes in user data.

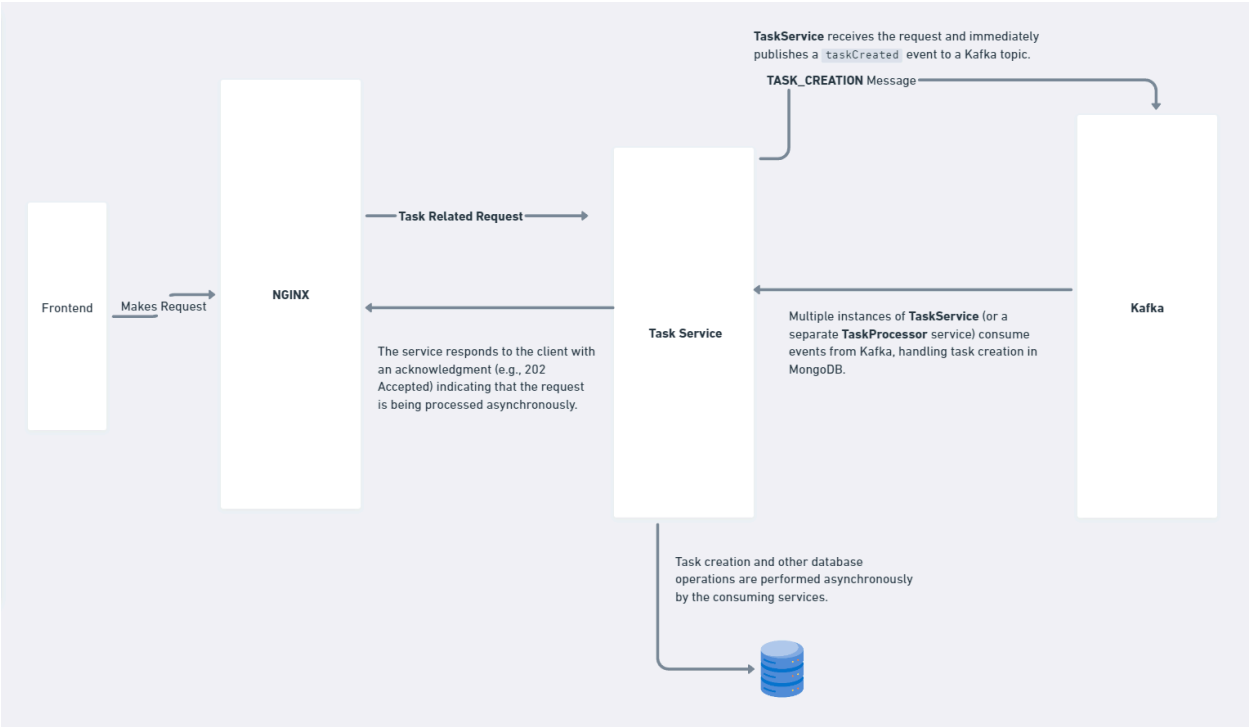
---

5. Scalability and Production Readiness

1. **Horizontal Scaling:**
    - **Microservices:** Deploy multiple instances of each microservice behind a load balancer.

    - **Kafka Consumers:** Utilize multiple instances of Kafka consumers for efficient message handling.
2. **Database Sharding:**
    - Implement sharding in MongoDB to distribute data across multiple instances, alleviating pressure from a single database.
3. **Asynchronous Processing:**
    - Use Kafka to facilitate asynchronous communication, decoupling request handling from processing logic to enhance performance under load.
4. **Caching:**
    - Employ caching strategies (e.g., Redis) for frequently accessed data to reduce database load and enhance response times.
5. **Health Monitoring and Alerts:**
    - Utilize monitoring tools (e.g., Prometheus, Grafana) to track service health and performance, establishing alerts for critical issues.
6. **Error Handling and Retries:**
    - Build resilient error handling and retry mechanisms for Kafka consumers to manage transient failures effectively.
7. **Security:**
    - Enforce secure communication (e.g., HTTPS) and implement strong authentication and authorization to safeguard sensitive data.
8. **Logging and Auditing:**
    - Adopt centralized logging solutions (e.g., ELK stack) and maintain thorough audit logs for change tracking and compliance.


Task Related
DB operations
Task Service Listen For
TASK_CREATION and
Will create Task
Accordingly
Task Creation Handeling
Task Relation Request
Task Service
TASK_CREATION Message
Frontend
NGINX
Comment Related Request
Comment Service
Message Queue
Kafka
Attachement Related Request
Attachement Service
User Related Request
User Service


TaskService receives the request and immediately
publishes a taskCreated event to a Kafka topic.
TASK_CREATION Message
Frontend
Makes Request
NGINX
Task Related Request
Task Service
Kafka
Multiple instances of TaskService (or a
separate TaskProcessor service) consume
events from Kafka, handling task creation in
MongoDB.
The service responds to the client with
an acknowledgment (e.g., 202
Accepted) indicating that the request
is being processed asynchronously.
Task creation and other database
operations are performed asynchronously
by the consuming services.