Calculator Microservices System with NestJS

Grok

July 14, 2025

1 Overview

This document describes a calculator system implemented using NestJS microservices. The system consists of a gateway microservice that receives client requests and four processing microservices, each handling a specific mathematical operation: addition (sum), multiplication, subtraction, and division. Each microservice is independent, communicates via TCP, and follows a modular design to demonstrate microservice architecture principles.

2 System Architecture

The system is divided into the following components:

- **Gateway Microservice** (127.0.0.1:3001): Acts as the entry point for client requests. It receives messages with an operation (e.g., sum, multiply) and numbers, then forwards them to the appropriate processing microservice.
- **Sum Microservice** (127.0.0.1:3002): Handles addition of numbers.
- **Multiply Microservice** (127.0.0.1:3003): Handles multiplication of numbers.
- **Subtract Microservice** (127.0.0.1:3004): Handles subtraction of numbers.
- **Divide Microservice** (127.0.0.1:3005): Handles division of numbers.
- **Test Client**: A script that sends test requests to the gateway to verify functionality.

The communication flow is as follows:

1. A client sends a message to the gateway with the pattern `{ cmd: 'calculate', operation: string, numbers: number[] }`.
2. The gateway routes the request to the appropriate processing microservice based on the operation.
3. The processing microservice performs the calculation and returns the result.
4. The gateway forwards the result back to the client.

3 Implementation

Each microservice is a NestJS application using TCP transport. Below is the structure and key code for each component.

3.1 Processing Microservices

Each processing microservice (`sum`, `multiply`, `subtract`, `divide`) has a similar structure but operates on a unique port and handles one operation. Below is an example for the Sum Microservice (127.0.0.1:3002).

3.1.1 main.ts

```
import { NestFactory } from '@nestjs/core';
import { Transport, MicroserviceOptions } from '@nestjs/microservices';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.createMicroservice<MicroserviceOptions>(
    AppModule,
    {
      transport: Transport.TCP,
      options: {
        host: '127.0.0.1',
        port: 3002,
      },
    },
  );
  await app.listen();
  console.log('Sum microservice is listening on port 3002');
}
bootstrap();
```

3.1.2 app.service.ts

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class AppService {
  sum(data: number[]): number {
    if (!data || data.length === 0) throw new Error('Input must be a non-empty array of numbers');
    return data.reduce((a, b) => a + b, 0);
  }
}
```

3.1.3 app.controller.ts

```
import { Controller } from '@nestjs/common';
import { MessagePattern } from '@nestjs/microservices';
import { AppService } from './app.service';

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @MessagePattern({ cmd: 'sum' })
  sum(data: number[]): number {
    return this.appService.sum(data);
  }
}
```

The other processing microservices (`multiply`, `subtract`, `divide`) follow the same structure but use ports 3003, 3004, and 3005, respectively, and implement their specific operations. For example, the Divide Microservice includes validation to prevent division by zero:

```
divide(data: number[]): number {
  if (!data || data.length < 2) throw new Error('Input must be an array with at least two numbers');
  if (data.slice(1).includes(0)) throw new Error('Division by zero is not allowed');
  return data.reduce((a, b) => a / b);
}
```

3.2 Gateway Microservice

The gateway microservice (127.0.0.1:3001) routes client requests to the appropriate processing microservice.

3.2.1 main.ts

```
import { NestFactory } from '@nestjs/core';
import { Transport, MicroserviceOptions } from '@nestjs/microservices';
import { AppModule } from './app.module';
import { AppService } from './app.service';

async function bootstrap() {
  const app = await NestFactory.createMicroservice<MicroserviceOptions>(
    AppModule,
    {
      transport: Transport.TCP,
      options: {
        host: '127.0.0.1',
        port: 3001,
      },
    },
  );

  const appService = app.get(AppService);
  await appService.connectToProcessingServices();

  await app.listen();
  console.log('Gateway microservice is listening on port 3001');
}
bootstrap();
```

3.2.2 app.service.ts

```
import { Injectable, OnModuleInit, OnModuleDestroy } from '
   @nestjs/common';
import { ClientProxy, ClientProxyFactory, Transport } from '
   @nestjs/microservices';

@Injectable()
export class AppService implements OnModuleInit, OnModuleDestroy
   {
  private clients: { [key: string]: ClientProxy } = {};

  constructor() {
    const services = [
      { operation: 'sum', port: 3002 },
      { operation: 'multiply', port: 3003 },
      { operation: 'subtract', port: 3004 },
      { operation: 'divide', port: 3005 },
    ];

    services.forEach(({ operation, port }) => {
      this.clients[operation] = ClientProxyFactory.create({
        transport: Transport.TCP,
        options: {
          host: '127.0.0.1',
          port,
        },
      });
    });
  }

  async onModuleInit() {
    await Promise.all(
      Object.keys(this.clients).map(operation =>
        this.connectWithRetry(operation, this.clients[operation])
           ,
      ),
    );
  }

  async connectWithRetry(operation: string, client: ClientProxy) {
    let retries = 5;
    while (retries > 0) {
      try {
        await client.connect();
        console.log(`Connected to ${operation} microservice`);
        return;
      } catch (error) {
        console.warn(`Connection to ${operation} failed, retrying ... (${retries} attempts left)`);
        retries--;
        if (retries === 0) throw error;
        await new Promise(resolve => setTimeout(resolve, 1000));
      }
    }
  }

  async calculate(operation: string, data: number[]): Promise<number> {
    const client = this.clients[operation];
    if (!client) {
      throw new Error(`Unsupported operation: ${operation}`);
    }
    try {
      const pattern = { cmd: operation };
      const result = await client.send<number>(pattern, data).toPromise();
      return result;
    } catch (error) {
      throw new Error(`Calculation failed for ${operation}: ${error.message}`);
    }
  }

  async onModuleDestroy() {
    for (const operation in this.clients) {
      if (this.clients[operation]) {
        this.clients[operation].close();
        console.log(`Connection to ${operation} microservice closed`);
      }
    }
  }
}
```

3.2.3 app.controller.ts

```
import { Controller } from '@nestjs/common';
import { MessagePattern } from '@nestjs/microservices';
import { AppService } from './app.service';

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @MessagePattern({ cmd: 'calculate' })
  async calculate(data: { operation: string; numbers: number[] })
     : Promise<number> {
    return this.appService.calculate(data.operation, data.numbers
       );
  }
}
```

4 Setup Steps

To set up and run the calculator system, follow these steps:

4.1 Step 1: Create Microservice Projects

Create five separate NestJS projects:

- `sum-microservice`
- `multiply-microservice`
- `subtract-microservice`
- `divide-microservice`
- `gateway-microservice`

For each, run:

```
nest new <service-name>
cd <service-name>
npm install @nestjs/core @nestjs/microservices rxjs
```

Replace the generated files with the provided `main.ts`, `app.module.ts`, `app.service.ts`, and `app.controller.ts` for each microservice.

4.2 Step 2: Start Processing Microservices

Start each processing microservice in its own terminal:

```
cd sum-microservice
npm run start
```

```
cd multiply-microservice
npm run start
```

```
cd subtract-microservice
npm run start
```

```
cd divide-microservice
npm run start
```

Verify logs:

- `Sum microservice is listening on port 3002`
- `Multiply microservice is listening on port 3003`
- `Subtract microservice is listening on port 3004`
- `Divide microservice is listening on port 3005`

4.3 Step 3: Start Gateway Microservice

Start the gateway microservice:

```
cd gateway-microservice
npm run start
```

Verify log: `Gateway microservice is listening on port 3001`.

4.4 Step 4: Test the System

Create a test client script (`test-client.ts`) to send requests to the gateway:

```
import { ClientProxyFactory, Transport, ClientProxy } from '
   @nestjs/microservices';

async function testCalculator() {
  let client: ClientProxy;
  try {
    client = ClientProxyFactory.create({
      transport: Transport.TCP,
      options: {
        host: '127.0.0.1',
        port: 3001,
      },
    });

    let retries = 5;
    while (retries > 0) {
      try {
        await client.connect();
        console.log('Connected to gateway microservice');
        break;
      } catch (error) {
        console.warn(`Connection failed, retrying... (${retries}
           attempts left)`);
        retries--;
        if (retries === 0) throw error;
        await new Promise(resolve => setTimeout(resolve, 1000));
      }
    }

    const operations = [
      { operation: 'sum', numbers: [1, 2, 3, 4] },
      { operation: 'multiply', numbers: [2, 3, 4] },
      { operation: 'subtract', numbers: [10, 3, 2] },
      { operation: 'divide', numbers: [100, 5, 2] },
    ];

    for (const op of operations) {
      try {
        const result = await client
          .send<number>({ cmd: 'calculate' }, op)
          .toPromise();
        console.log(`Result of ${op.operation}(${op.numbers}): ${result}`);
      } catch (error) {
        console.error(`Error for ${op.operation}: ${error.message}`);
      }
    }
  } catch (error) {
    console.error('Failed to communicate with gateway:', error.message);
  } finally {
    if (client) client.close();
  }
}

testCalculator();
```

Install dependencies and run:

```
npm install @nestjs/microservices rxjs
npx tsc test-client.ts
node test-client.js
```

Expected output:

```
Connected to gateway microservice
Result of sum([1,2,3,4]): 10
Result of multiply([2,3,4]): 24
Result of subtract([10,3,2]): 5
Result of divide([100,5,2]): 10
```

5 Troubleshooting

If you encounter issues such as `ECONNREFUSED`:

- **Verify Microservices**: Ensure all microservices are running and listening on their respective ports (3001–3005). Check with:

```
    netstat -tuln | grep 300[1-5]
```

- **Start Order**: Start processing microservices first, then the gateway, then the test client.
- **Port Conflicts**: If a port is in use, change it in the respective `main.ts` file.
- **Firewall**: Allow TCP ports 3001–3005:

```
    sudo ufw allow 3001:3005/tcp
```

- **Dependencies**: Ensure all projects have:

```
    npm install @nestjs/core @nestjs/microservices rxjs
```

6 Next Steps

- **Add Validation**: Use `ValidationPipe` with `class-validator` to validate inputs.
- **Logging**: Implement a logging microservice using `@EventPattern`.
- **Other Transports**: Switch to Redis or NATS for production-grade communication.
- **Testing**: Write unit tests using `@nestjs/testing`.