PENGEMBANGAN WEB (TEORI)

LAPORAN EKSPERIMEN MENGENAI PERBANDINGAN PERFORMA ANTARA NODE.JS VS. JAVA (SPRING BOOT) PADA MICROSERVIS

Laporan ini disusun untuk memenuhi tugas 1 mata kuliah Pengembangan Web (Teori)

Disusun oleh kelompok B4:

Asri Husnul Rosadi	221524035
Faris Abulkhoir	221524040
Mahardika Pratama	221524044
Muhamad Fahri Yuwan	221524047
Najib Alimudin Fajri	221524053
Septyana Agustina	221524058
Sarah	221524059

Dosen Pengampu:
Joe Lian Min, M.Eng.

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
PROGRAM STUDI D4 TEKNIK INFORMATIKA
POLITEKNIK NEGERI BANDUNG

2024

DAFTAR ISI

A. **IDENTIFIKASI PROBLEM**

Dalam konteks pengembangan aplikasi berbasis microservices, pemilihan teknologi yang tepat untuk backend dapat mempengaruhi performa, skalabilitas, dan efisiensi keseluruhan sistem. Node.js dan Java (Spring Boot) adalah dua pilihan populer untuk membangun microservices, masing-masing dengan kelebihan dan kekurangan. Node.js, dengan arsitektur berbasis event-driven dan non-blocking I/O, menawarkan performa tinggi dalam menangani banyak koneksi simultan dan operasional I/O. Di sisi lain, Spring Boot menggunakan pendekatan berbasis thread yang stabil dan memiliki ekosistem yang matang serta dukungan kuat untuk berbagai jenis database dan integrasi. Masalah yang perlu diidentifikasi adalah perbedaan performa dan efisiensi antara kedua teknologi ini ketika diterapkan dalam lingkungan microservices yang membutuhkan skalabilitas tinggi, pemrosesan data intensif, dan kemampuan untuk menangani beban tinggi.

B. **DESKRIPSI PROBLEM**

Permasalahan utama dalam membandingkan Node.js dengan Java (Spring Boot) dalam konteks microservices adalah bagaimana kedua platform tersebut menangani skenario beban tinggi dan pengelolaan koneksi simultan. Node.js, dengan pendekatan asinkron dan event-driven-nya, mungkin menunjukkan keunggulan dalam hal latensi rendah dan kemampuan untuk menangani banyak permintaan secara bersamaan. Namun, kemampuan ini juga tergantung pada bagaimana aplikasi dioptimalkan untuk skala besar. Sementara itu, Spring Boot, yang menggunakan model pemrograman berbasis thread, mungkin menawarkan stabilitas dan manajemen sumber daya yang lebih baik dalam lingkungan yang lebih tradisional namun dengan overhead yang mungkin lebih tinggi dalam hal penggunaan memori dan latency. Perbandingan ini perlu dilakukan untuk mengevaluasi kelebihan dan kekurangan masing-masing teknologi dalam hal throughput, latency, dan skalabilitas, serta bagaimana masing-masing platform mengatasi berbagai beban dan tuntutan dalam konteks microservices.

C. **METODOLOGI EKSPERIMEN**

1) Tujuan

   Tujuan dari eksperimen ini adalah untuk membandingkan performa dan efisiensi antara Node.js dan Java (Spring Boot) dalam konteks aplikasi microservices. Eksperimen ini bertujuan untuk menilai bagaimana kedua teknologi ini menangani

beban tinggi, latensi, throughput, dan skalabilitas ketika diterapkan pada arsitektur microservices. Hasil dari eksperimen ini akan memberikan wawasan tentang kelebihan dan kekurangan masing-masing teknologi dalam skenario nyata.

2) Desain Eksperimen

Eksperimen ini akan melibatkan pengembangan dua layanan microservices yang identik: satu menggunakan Node.js dan satu lagi menggunakan Java (Spring Boot). Kedua layanan akan diimplementasikan untuk menyediakan endpoint API yang serupa. Uji beban akan dilakukan untuk mensimulasikan kondisi penggunaan yang berat, dan analisis performa akan dilakukan untuk mengukur waktu respon, throughput, dan penggunaan sumber daya. Layanan akan dideploy dalam lingkungan yang sama (misalnya, menggunakan container Docker untuk memastikan konsistensi) dan akan diuji dengan alat yang sama untuk memastikan validitas hasil.

3) Variabel Eksperimen

- Variabel Independen: Teknologi yang digunakan (Node.js vs. Java (Spring Boot)).
- Variabel Terikat:
    - Waktu respon (latency)
    - Throughput (jumlah permintaan yang diproses per detik)
    - Penggunaan CPU
    - Penggunaan memori
    - Skalabilitas (kemampuan untuk menangani beban yang meningkat).
- Variabel Kontrol:
    - Lingkungan pengujian (server yang sama, konfigurasi jaringan yang sama)
    - Alat pengujian (misalnya, Apache JMeter untuk load testing)
    - Beban pengujian (jumlah permintaan simultan, ukuran data yang diproses)

4) Langkah-Langkah Pengujian

- Pengembangan Aplikasi:
    - Set up lingkungan pengujian dengan container Docker untuk Node.js dan Java (Spring Boot).

- Konfigurasi database dan integrasi yang diperlukan untuk kedua layanan.
- Instalasi alat pengujian seperti Apache JMeter untuk load testing.

- Pengembangan Aplikasi:
  - Kembangkan dua layanan microservices dengan endpoint API yang identik: satu menggunakan Node.js dan satu lagi menggunakan Java (Spring Boot).
  - Pastikan kedua layanan memiliki fitur yang sama untuk memproses data dan menghasilkan respons yang serupa.
- Pengujian Beban:
  - Lakukan uji beban dengan menggunakan alat pengujian untuk mensimulasikan berbagai tingkat beban pengguna (misalnya, 100, 500, 1000 permintaan per detik).
  - Catat metrik performa selama pengujian, termasuk waktu respon, throughput, penggunaan CPU, dan penggunaan memori.
- Pengujian Skalabilitas:
  - Uji bagaimana masing-masing layanan mengatasi peningkatan beban dengan menambah jumlah permintaan secara bertahap.
  - Ukur perubahan dalam performa seiring dengan penambahan beban.
- Pengumpulan Data dan Analisis:
  - Kumpulkan data dari alat pengujian mengenai waktu respon, throughput, dan penggunaan sumber daya.
  - Analisis hasil untuk menentukan perbedaan dalam performa dan efisiensi antara Node.js dan Java (Spring Boot).

5) Analisis Hasil:

- Waktu Respon: Bandingkan waktu respon rata-rata dari kedua layanan untuk menentukan teknologi mana yang memiliki latensi lebih rendah.
- Throughput: Evaluasi throughput untuk mengidentifikasi teknologi mana yang dapat memproses lebih banyak permintaan per detik.

- Penggunaan CPU dan Memori: Bandingkan penggunaan CPU dan memori untuk menilai efisiensi sumber daya dari masing-masing teknologi.
- Skalabilitas: Analisis bagaimana performa masing-masing teknologi berubah seiring dengan peningkatan beban untuk menilai kemampuan skalabilitas.
- Kesimpulan: Berdasarkan hasil analisis, buat kesimpulan tentang kelebihan dan kekurangan masing-masing teknologi dalam konteks aplikasi microservices. Pertimbangkan faktor-faktor seperti latensi, throughput, efisiensi sumber daya, dan kemampuan untuk menangani beban tinggi dalam rekomendasi akhir..

D. PELAKSANAAN EKSPERIMEN

Node.js

1) Setup dan Konfigurasi:

   Instalasi Node.js dan NPM. Pastikan Node.js dan npm (Node Package Manager) terinstal di sistem Anda. Jika belum, unduh dan instal dari situs resmi Node.js.

   Membuat Proyek Node.js

   a. Inisialisasi Proyek:

```
mkdir node-microservice
cd node-microservice
npm init -y
```

   b. Install Depedencies

```
npm install express body-parser
```

2) Pengembangan Aplikasi
   Buat file server.js di dalam direktori proyek dengan isi berikut:

```
const express = require('express');
const bodyParser = require('body-parser');

// Inisialisasi aplikasi Express
const app = express();
const port = 3000;

// Middleware untuk parsing JSON
app.use(bodyParser.json());

// Endpoint API untuk mendapatkan data
app.get('/api/data', (req, res) => {
```

```
    res.json({ message: 'Hello from Node.js microservice!'
});
});

// Endpoint API untuk menerima data
app.post('/api/data', (req, res) => {
    const data = req.body;
    res.json({ received: data });
});

// Jalankan server
app.listen(port, () => {
    console.log(`Server berjalan di
http://localhost:${port}`);
});
```

3) Menjalankan Layanan
   a. Jalankan server dengan perintah:

```
node server.js
```

   b. Buka browser atau gunakan alat seperti `curl` untuk memverifikasi endpoint:

```
curl http://localhost:3000/api/data
```

   c. Anda harus menerima respons:

```
{ "message": "Hello from Node.js microservice!" }
```

4) Pengujian Beban dengan Apache-Jmeter
   Instalasi Apache JMeter
   Unduh dan instal Apache JMeter dari situs resmi Apache.
   Membuat Tes Beban
   a. Buka JMeter: Jalankan JMeter dengan menjalankan jmeter.bat (Windows) atau jmeter (Linux/Mac) dari direktori bin JMeter.
   b. Konfigurasi Tes:
      - Menambahkan Thread Group:
        - Klik kanan pada "Test Plan" > "Add" > "Threads (Users)" > "Thread Group".
        - Set jumlah thread (misalnya, 100), ramp-up period (misalnya, 10 detik), dan loop count (misalnya, 1).
      - Menambahkan HTTP Request Sampler:
        - Klik kanan pada "Thread Group" > "Add" > "Sampler" > "HTTP Request".
        - Set "Server Name or IP" ke localhost dan "Port Number" ke 3000.

        - Set "Path" ke /api/data dan "Method" ke GET atau POST sesuai kebutuhan.
        - Untuk POST, tambahkan parameter dalam tab "Body Data".
    c. Menambahkan Listener untuk Melihat Hasil:
    Klik kanan pada "Thread Group" > "Add" > "Listener" > "View Results Tree" atau "Summary Report".
    d. Menjalankan Tes:
        - Klik ikon start (berwarna hijau) di toolbar untuk menjalankan tes.
        - Monitor hasil pengujian di listener yang telah ditambahkan.

Jawa Springboot

1) Setup dan Konfigurasi:

a. Instalasi JDK dan Maven

Pastikan JDK (Java Development Kit) dan Maven terinstal di sistem Anda. Jika belum, unduh dan instal dari situs resmi Oracle untuk JDK dan situs resmi Maven untuk Maven.

b. Membuat Proyek Spring Boot

Menggunakan Spring Initializr

- Buka Spring Initializr: Kunjungi Spring Initializr.
- Konfigurasi Proyek:
    - Project: Maven Project
    - Language: Java
    - Spring Boot: Pilih versi terbaru
    - Group: com.example
    - Artifact: springboot-microservice
    - Name: springboot-microservice
    - Packaging: Jar
    - Java: Versi yang sesuai dengan JDK yang Anda gunakan
    - Dependencies: Tambahkan "Spring Web"
- Generate dan Unduh: Klik "Generate" untuk mendownload file ZIP proyek.
- Ekstrak dan Buka Proyek: Ekstrak file ZIP dan buka proyek di IDE seperti IntelliJ IDEA atau Eclipse.

2) Pengembangan Aplikasi
Buka src/main/java/com/example/springbootmicroservice/SpringbootMicroserviceApplication.java dan tambahkan kode berikut:

```
package com.example.springbootmicroservice;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringbootMicroserviceApplication {
    public static void main(String[] args) {

SpringApplication.run(SpringbootMicroserviceApplication.class
, args);
    }
}
```

Buat file DataController.java di src/main/java/com/example/springbootmicroservice/ dengan isi berikut:

```
package com.example.springbootmicroservice;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
public class DataController {

    @GetMapping("/data")
    public String getData() {
        return "{\"message\": \"Hello from Spring Boot
microservice!\"}";
    }

    @PostMapping("/data")
    public String postData(@RequestBody String data) {
        return "{\"received\": " + data + "}";
    }
}
```

Buka src/main/resources/application.properties dan tambahkan konfigurasi berikut jika diperlukan (misalnya, port server):

```
server.port=8080
```

3) Menjalankan Layanan
   a. Jalankan aplikasi dengan perintah:

```
mvn spring-boot:run
```

b. Buka browser atau gunakan alat seperti `curl` untuk memverifikasi endpoint:

```
curl http://localhost:8080/api/data
```

c. Anda harus menerima respons:

```
{ "message": "Hello from Spring Boot microservice!" }
```

4) Pengujian Beban dengan Apache-Jmeter
Instalasi Apache JMeter
Unduh dan instal Apache JMeter dari situs resmi Apache.
Membuat Tes Beban
   a. Buka JMeter: Jalankan JMeter dengan menjalankan jmeter.bat (Windows) atau jmeter (Linux/Mac) dari direktori bin JMeter.
   b. Konfigurasi Tes:
      - Menambahkan Thread Group:
        - Klik kanan pada "Test Plan" > "Add" > "Threads (Users)" > "Thread Group".
        - Set jumlah thread (misalnya, 100), ramp-up period (misalnya, 10 detik), dan loop count (misalnya, 1).
      - Menambahkan HTTP Request Sampler:
        - Klik kanan pada "Thread Group" > "Add" > "Sampler" > "HTTP Request".
        - Set "Server Name or IP" ke localhost dan "Port Number" ke 8080.
        - Set "Path" ke /api/data dan "Method" ke GET atau POST sesuai kebutuhan.
        - Untuk POST, tambahkan parameter dalam tab "Body Data".
   c. Menambahkan Listener untuk Melihat Hasil:
      Klik kanan pada "Thread Group" > "Add" > "Listener" > "View Results Tree" atau "Summary Report".
   d. Menjalankan Tes:
      - Klik ikon start (berwarna hijau) di toolbar untuk menjalankan tes.
      - Monitor hasil pengujian di listener yang telah ditambahkan.

E. ANALISIS HASIL EKSPERIMEN

Node.js	Jawa Springboot

Thread Name:Thread Group 1-3 Sample Start:2024-09-06 09:03:19 WIB Load time:114 Connect Time:77 Latency:103 Size in bytes:281 Sent bytes:124 Headers size in bytes:235 Body size in bytes:46 Sample Count:1 Error Count:0 Data type ("text"|"bin"|""):text Response code:200 Response message:OK	Thread Name:Thread Group 2-1 Sample Start:2024-09-06 09:09:22 WIB Load time:15 Connect Time:2 Latency:15 Size in bytes:213 Sent bytes:124 Headers size in bytes:162 Body size in bytes:51 Sample Count:1 Error Count:0 Data type ("text"|"bin"|""):text Response code:200 Response message:

Berdasarkan hasil pengujian menggunakan Apache JMeter, performa layanan microservices antara Node.js dan Spring Boot menunjukkan perbedaan yang signifikan. Untuk Node.js microservice, waktu muat (Load time) tercatat sebesar 114 ms, dengan waktu koneksi (Connect Time) 77 ms dan latensi sebesar 103 ms. Hasil ini menunjukkan bahwa sebagian besar waktu dihabiskan untuk membangun koneksi dan menunggu respons. Ukuran respons sebesar 281 byte dengan ukuran tubuh (Body size) sebesar 46 byte, dan tidak ada error yang terjadi selama pengujian.

Sementara itu, Spring Boot microservice menunjukkan performa yang lebih baik dengan waktu muat hanya 15 ms, waktu koneksi 2 ms, dan latensi 15 ms. Ukuran respons tercatat sebesar 213 byte dengan ukuran tubuh 51 byte. Kedua microservices berhasil merespons dengan kode status 200 tanpa adanya error, menunjukkan bahwa keduanya dapat berfungsi dengan baik di bawah kondisi pengujian ini.

Secara keseluruhan, Spring Boot memberikan waktu respons yang lebih cepat dan latensi yang lebih rendah dibandingkan dengan Node.js, yang menunjukkan efisiensi yang lebih baik dalam menangani permintaan dengan overhead yang lebih minimal. Namun, analisis lebih lanjut diperlukan untuk memahami faktor-faktor yang mempengaruhi perbedaan ini, seperti konfigurasi server dan optimalisasi kode pada masing-masing platform.

F. **KESIMPULAN**

Kesimpulan dari praktikum ini menunjukkan bahwa terdapat perbedaan signifikan dalam performa antara microservices yang dibangun menggunakan Node.js dan Spring Boot. Berdasarkan hasil pengujian menggunakan Apache JMeter, Spring Boot memiliki waktu respons yang lebih cepat dengan latensi yang lebih rendah dibandingkan dengan Node.js. Spring Boot menunjukkan waktu muat yang lebih singkat (15 ms) dan latensi yang lebih rendah (15 ms), sedangkan Node.js menunjukkan waktu muat yang lebih tinggi (114 ms) dan latensi yang lebih lama (103 ms). Kedua layanan berhasil menangani permintaan tanpa error, namun Spring Boot secara konsisten memberikan performa yang lebih efisien.

Hasil ini mengindikasikan bahwa Spring Boot lebih unggul dalam menangani beban dan responsivitas dalam skenario pengujian ini. Namun, perlu diperhatikan bahwa hasil ini bisa dipengaruhi oleh berbagai faktor seperti konfigurasi server, optimasi kode, dan kondisi jaringan. Oleh karena itu, pilihan teknologi terbaik tetap harus disesuaikan dengan kebutuhan spesifik proyek, serta mempertimbangkan faktor lain seperti skalabilitas, kemudahan pengembangan, dan pemeliharaan. Praktikum ini memberikan wawasan penting bagi pengembang dalam memilih teknologi yang tepat untuk membangun microservices yang optimal.