**CLIENT SIDE DISCOVERY**

Client-side discovery dalam arsitektur microservice adalah pola di mana klien (client) bertanggung jawab untuk menemukan lokasi layanan mikro (microservices) yang ingin diaksesnya. Berikut penjelasan lebih lanjut tentang konsep ini:

**Cara Kerja Client-Side Discovery**

1. **Service Registry**:
   * Semua layanan mikro mendaftarkan alamat dan metadata mereka ke dalam service registry. Registry ini bertindak sebagai direktori yang menyimpan informasi tentang semua layanan yang tersedia.
2. **Client**:
   * Ketika klien ingin mengakses layanan tertentu, ia akan mengambil daftar instance layanan dari service registry.
   * Klien kemudian menggunakan informasi ini untuk melakukan load balancing dan memutuskan instance mana yang akan diakses.
3. **Load Balancing**:
   * Klien melakukan load balancing secara mandiri berdasarkan informasi yang didapat dari service registry. Ini bisa melibatkan algoritma seperti round-robin, random, atau yang lebih kompleks.

**Keuntungan Client-Side Discovery**

* **Efisiensi**: Karena klien yang melakukan load balancing, ini bisa lebih efisien dan responsif terhadap perubahan dalam topologi layanan.
* **Skalabilitas**: Mengurangi beban pada load balancer terpusat dan dapat mendistribusikan beban secara lebih merata.
* **Fleksibilitas**: Klien dapat menerapkan strategi load balancing dan penemuan layanan yang disesuaikan dengan kebutuhan spesifik mereka.

**Kekurangan Client-Side Discovery**

* **Kompleksitas di Klien**: Menambahkan logika penemuan layanan dan load balancing ke klien dapat meningkatkan kompleksitas kode di sisi klien.
* **Ketergantungan pada Service Registry**: Keberhasilan client-side discovery bergantung pada ketersediaan dan konsistensi service registry. Jika registry mengalami masalah, penemuan layanan juga akan terpengaruh.

**Contoh Implementasi**

* **Netflix Eureka**: Sebuah service registry dan client library yang sering digunakan dalam arsitektur microservice.
* **Consul**: Sebuah solusi yang lebih luas untuk service discovery, konfigurasi, dan segmentasi layanan.

Client-side discovery adalah salah satu pendekatan dalam arsitektur microservice yang membantu meningkatkan fleksibilitas dan skalabilitas, meskipun membutuhkan manajemen yang lebih kompleks di sisi klien.

**SERVICE REGISTRY**

Service registry adalah komponen penting dalam arsitektur microservice yang berfungsi sebagai direktori atau katalog di mana semua layanan mikro mendaftarkan keberadaannya dan metadata terkait. Registry ini memungkinkan layanan untuk menemukan satu sama lain tanpa perlu mengetahui alamat fisik layanan lain secara langsung.

**Fungsi Utama Service Registry**

1. **Pendaftaran Layanan (Service Registration)**:
   * Setiap layanan mikro mendaftarkan alamatnya (misalnya, URL atau IP dan port) serta informasi tambahan (seperti metadata, versi, status) ke dalam service registry.
   * Pendaftaran ini bisa dilakukan secara manual oleh pengembang atau otomatis oleh layanan itu sendiri pada saat startup.
2. **Penemuan Layanan (Service Discovery)**:
   * Layanan mikro atau klien dapat mengakses service registry untuk menemukan alamat layanan lain yang ingin mereka panggil.
   * Registry akan memberikan daftar instance layanan yang tersedia sehingga klien dapat memilih salah satu untuk diakses.
3. **Pemantauan Kesehatan (Health Monitoring)**:
   * Service registry sering dilengkapi dengan kemampuan untuk memantau kesehatan layanan yang terdaftar.
   * Registry secara berkala memeriksa apakah layanan masih aktif dan sehat. Jika tidak, registry akan memperbarui status layanan tersebut atau menghapusnya dari daftar layanan yang tersedia.

**Keuntungan Menggunakan Service Registry**

* **Dekoupling**: Memungkinkan layanan mikro untuk berkomunikasi tanpa perlu mengetahui lokasi satu sama lain sebelumnya, mengurangi ketergantungan dan meningkatkan fleksibilitas.
* **Load Balancing**: Memudahkan implementasi load balancing karena klien dapat memilih dari beberapa instance layanan yang terdaftar.
* **Skalabilitas**: Mendukung penskalaan layanan dengan menambahkan atau menghapus instance secara dinamis tanpa memerlukan konfigurasi ulang secara manual.

**Contoh Implementasi Service Registry**

1. **Netflix Eureka**:
   * Sering digunakan dalam ekosistem Spring Cloud untuk pendaftaran dan penemuan layanan. Eureka adalah bagian dari stack Netflix OSS.
   * Mendukung pemantauan kesehatan layanan dan memungkinkan registrasi otomatis.
2. **Consul**:
   * Menyediakan pendaftaran layanan, penemuan layanan, dan juga kemampuan konfigurasi dan segmentasi jaringan.
   * Mendukung fitur-fitur canggih seperti health checks dan key-value store untuk konfigurasi.
3. **Apache Zookeeper**:
   * Digunakan sebagai service registry dalam beberapa sistem, terutama yang memerlukan konsistensi data yang tinggi.
   * Sering digunakan dengan Hadoop, HBase, dan Kafka.
4. **Etcd**:
   * Sebuah store key-value distribusi yang sering digunakan untuk menyimpan data konfigurasi dan pendaftaran layanan dalam sistem Kubernetes.

**Proses Kerja Service Registry**

1. **Layanan Mendaftar ke Registry**:
   * Pada saat startup, layanan mengirimkan informasi pendaftarannya ke service registry.
2. **Klien Mengakses Registry untuk Menemukan Layanan**:
   * Klien mengirimkan permintaan ke registry untuk mendapatkan daftar instance layanan yang tersedia.
3. **Registry Menyediakan Informasi Layanan**:
   * Registry merespon dengan daftar instance yang memenuhi kriteria pencarian klien.
4. **Load Balancing dan Health Checks**:
   * Klien menggunakan informasi ini untuk load balancing dan registry terus memantau kesehatan layanan yang terdaftar.

Dengan menggunakan service registry, arsitektur microservice menjadi lebih dinamis, fleksibel, dan mampu menangani perubahan topologi jaringan dengan lebih baik.

**CENTRALIZED CONFIGURATION**

Centralized configuration dalam konteks arsitektur microservice adalah pendekatan di mana semua konfigurasi aplikasi disimpan dan dikelola di satu lokasi pusat. Ini memungkinkan semua layanan mikro (microservices) untuk mengakses dan mengelola konfigurasi mereka dari satu sumber, sehingga memudahkan pengelolaan, konsistensi, dan perubahan konfigurasi secara real-time.

**Keuntungan Centralized Configuration**

1. **Konsistensi**:
   * Semua layanan menggunakan satu sumber konfigurasi yang sama, memastikan bahwa konfigurasi tetap konsisten di seluruh layanan.
2. **Pengelolaan Mudah**:
   * Memungkinkan administrator untuk mengelola konfigurasi secara terpusat, mengurangi kerumitan pengelolaan konfigurasi di setiap layanan secara individu.
3. **Kemampuan Adaptasi**:
   * Memungkinkan perubahan konfigurasi diterapkan secara real-time tanpa perlu me-restart layanan, sehingga meningkatkan fleksibilitas dan kecepatan respons terhadap perubahan.
4. **Keamanan**:
   * Konfigurasi sensitif seperti kredensial dapat dienkripsi dan disimpan dengan aman di satu lokasi pusat.
5. **Versi dan Riwayat**:
   * Dapat menyimpan versi sebelumnya dari konfigurasi, memudahkan rollback jika ada masalah dengan konfigurasi terbaru.

**Contoh Implementasi Centralized Configuration**

1. **Spring Cloud Config**:
   * Framework yang menyediakan solusi centralized configuration untuk aplikasi Spring.
   * Mendukung konfigurasi yang diambil dari berbagai sumber seperti Git, SVN, dan file sistem lokal.
   * Konfigurasi dapat diakses melalui REST API yang disediakan oleh Spring Cloud Config Server.
2. **Consul**:
   * Selain sebagai service registry, Consul juga menyediakan fitur untuk centralized configuration.
   * Menggunakan key-value store untuk menyimpan konfigurasi yang dapat diakses oleh semua layanan.
3. **Etcd**:
   * Store key-value distribusi yang banyak digunakan dalam ekosistem Kubernetes untuk menyimpan konfigurasi dan metadata aplikasi.
   * Mendukung konfigurasi dinamis dan berbasis event.
4. **Zookeeper**:
   * Digunakan untuk penyimpanan konfigurasi dalam beberapa sistem besar yang memerlukan konsistensi data tinggi.
   * Mendukung watch pada perubahan data sehingga layanan dapat bereaksi terhadap perubahan konfigurasi.