Jawaban dari pertanyaan :

2. Sebuah external service sering:

- Timeout
- Return HTTP 500
- Kadang lambat tapi berhasil

Pertanyaan:

- **Dalam kondisi apa retry tidak boleh dilakukan?**
  1. Retry tidak boleh dilakukan di POST request , karena POST request itu termasuk operasi Non-idempotent, yang kalau tidak di tangani dengan baik requestnya bisa berulang kali di buat. Hal ini bisa diatasi dengan cara selalu menyematkan idempotent=key di header setiap POST request yang sensitive seperti process Payment, pembuatan tiket pesawat.

  2. Juga ketika terjadi 400 Bad Request(request format salah), 401 Unauthorized(authentication failed), 403 Forbidden (no permission), 404 Not Found (resource tidak di temukan), melakukan retry hanya akan akan menyia-nyiakan resources.

  3. Tidak adanya batasan retry.

- **Apa perbedaan retry dan circuit breaker?**
  Retry itu selalu melakukan pengulangan request berkali kali sampai berhasil. Sedangkan Circuit Breaker digunakan untuk menghentikan request kembali dikirimkan dengan syarat request sudah di retry sampai batas tertentu dan semuanya gagal, dengan cara itu service/ server punya waktu untuk recover.

- **Apa risiko retry tanpa jitter atau backoff?**
  Resiko retry tanpa Jitter atau backoff Adalah masalah yang sering disebut Thundering herd problem Dimana jika 1000 request gagal bersamaan, retry akan selalu terjadi berulang dengan 1000 request yang sama, hal ini bisa membuat system susah recover dan dapat menghabiskan resource seperti memory, thread dan connection pool.

3. 2Sistem terdiri dari 8 microservice. User melaporkan: “data saya hilang”, tetapi tidak ada error di log.

Pertanyaan:

- **Informasi minimum apa yang wajib ada di setiap log?**
  - Trace Id -> identitas unique dari event atau entri
  - Timestamp -> berisi waktu event atau request terjadi
  - Log level -> isinya ERROR, DEBUG, INFO, WARN, dan FATAL, yang bertujuan untuk mengidentifikasi jenis lognya
  - Message -> isi konten dari log, dapat berupa error details
  - Request Info -> berisi request method, path dan sataus
  - Service Name/ID -> memudahkan kita mencari tahu service apa yang generate lognya. (sangat berguna di microservice).

- **Bagaimana cara menelusuri satu request dari awal sampai akhir?**
  Salah satu cara yang dapat dilakukan adalah menggunakan distributed tracing dengan trace_idnya dimulai dari pretama request dibuat sampai ke akhir.

- **Apa dampak jika traceId tidak konsisten antar service?**
  Kita tidak bisa melakukan tracing service ke service karena traceId yang selalu berubah jika masuk ke service terkait karena susah mengaitkannya, mungkin masih bisa dikaitkan berdasarkan timestamp tapi bakal memakan waktu.

4. Kamu menggunakan message broker (Kafka / RabbitMQ) untuk event:

- CustomerUpdated

Kondisi:

- Ordering event penting
- Duplicate event tidak bisa dihindari

Pertanyaan:

- **Bagaimana memastikan consumer bersifat idempotent?**
  Pertama alasan consumer butuh bersifat idempotent karena ketika broker memiliki masalah jaringan message bisa dikirim berkali kali, jadi salah satu cara memastikan consumer bersifat idempotent yaitu pemakaian idempotent_key di setiap request atau message. Cara lain ialah version based idempotent, Dimana setiap request langsung mengubah value dari versi di database.
- **Apakah ordering dijamin oleh broker atau consumer?**
  Ordering dijamin oleh broker dan consumer walaupun dengan cara berbeda. Kalau broker menjamin ordering ketika melakukan pengiriman pesan, contohnya di kafka message masuk ke dalam partisi yang sama berurutan dan dikirimkan berurutan juga.

Tapi tidak berurutan jika salah satu data ada dipartisi yang berbeda.
Sedangkan di pihak consumer walaupun message datang bersamaan berurutan tapi jika diprocess secara parallel, contohnya thread 2 selesai di eksekusi duluan dari thread 1 urutannya akan berubah. Untuk mengatasi ini kita harus memastikan di consumer processnya di lakukan secara berurutan.

- **Apa dampaknya jika ordering tidak terjaga?**
  Dampaknya jika ordering tidak terjaga even lamabisa menimpa event terbaru, atau bahkan bisa menyebabkan data corruption. Lalu masalah system nya susah untuk ditrace.

5. Sebuah service memiliki traffic tinggi dan query database mahal.
Pertanyaan:

- **Kapan cache harus di-invalidate?**
  Kita bisa invalidate date di cache contohnya jika ada perubahan data di database utama, data sudah kadaluwarsa, data menjadi tidak relevan(seperti data stok barang realtime).
- **Apakah cache boleh dijadikan source of truth?**
  Tidak, karena sifat cache yang memiliki usia atau ada gangguan di cache tools dan pada akhirnya cache nanti bakal hilang atau di hapus. Untuk itu database harus tetap sebagai source of truth.
- **Bagaimana mencegah cache stampede?**
  Untuk mencegah cache stampede, kitab isa melakukan locking pada cachenya untuk memastikan hanya satu permintaan yang memperbarui cache yang kedaluwarsa. Ada juga cara lain seperti melakukan pembaharuan cache sebelum kadaluarsa secara otomatis terkhusus untuk cache yang serung dikunjungi.

6. Komunikasi antar microservice menggunakan HTTP.

Pertanyaan:

- **Apakah penggunaan API key sudah cukup?**
  Hanya mengggunakan API key saja tidak cukup karena dengan penggunaan API keys yang bersifat statikd dan selalu dikirimkan melalui header HTTP. Jika API keys bocor secara tidak sengaja sekali saja, siapa pun bisa menyalahgunakannya tanpa batas waktu. Ada beberapa solusi untuk mengatasinya contohnya Mutual TLS, OAuth2 dengan JWT, dans service mesh.
- **Bagaimana mencegah service palsu mengakses API internal?**
  Kita bisa menerapkan mutual TLS Dimana kedua ketika dua aplikasi saling berkomunikasi

kedua service harus saling menunjukkan sertifikat digital mereka, dan menolak akses jika sertifikat tidak valid

- **Apa risiko penggunaan JWT tanpa expiry pendek untuk komunikasi internal?**
  Resiko yang paling buruk ialah jika token dicuri , orang lain bisa menggunakannya dalam waktu yang lama . Untuk mengatasi ini jwt butuh token yang berumur pendek tapi bisa melakukan refresh token secara otomatis, mungkin bakal memakan sedikit resource tapi berguna di case ini.