Kegagalan dan Latensi Framework

Pertemuan 12

Konsep Circuit Breaker

- Ide dari pola circuit breaker itu relative simpel dan mirip dengan implementasi circuit breaker di dunia elektrikal dunia nyata. Ketika saklar dalam posisinya menyala (tertutup) maka arus listrik mengalir ke rangkaian rumah
- Tetapi jika terjadi kesalahan seperti korslet, maka circuit breaker akan terbuka menghentikan aliran listrik sebelum merusak perangkat elektronik

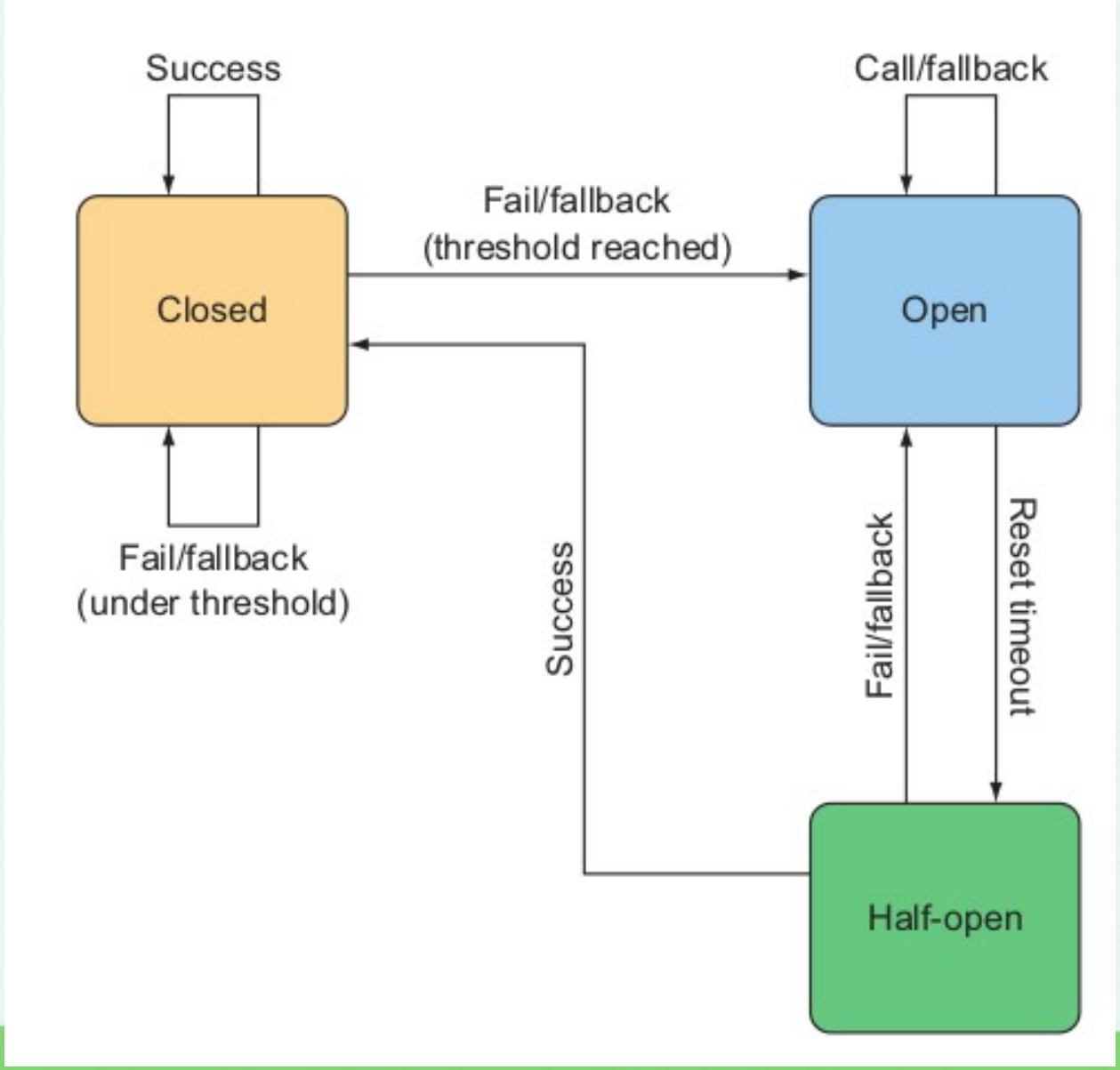
Lanjutan

- Circuit Breaker di dalam perangkat lunak dijalankan dalam posisi rangkaian terteutup yang mengizinkan eksekusi metode.
- Jika dalam keadaan tertentu ada metode yang gagal (melewati batas tetentu), maka rangkaian akan terbuka dan eksekusi dihentikan
- Selain dihentikan, perangkat lunak menyediakan metode cadangan dan self-correcting.

Lanjutan

- If the protected method fails within a given threshold of failure, then a fallback method can be called in its place. Once the circuit opens, that fallback method will be called almost exclusively
- Jika metode yang terlindungi gagal menurut batas aman, maka metode fallback bisa dipanggil. Ketika rangkaian terbuka, maka metode fallback akan dipanggil secara eksklusif

Ilustrasi


Success
Call/fallback
Fail/fallback
(threshold reached)
Closed
Open
Fail/fallback
(under threshold)
Success
Fail/fallback
Reset timeout
Half-open

3 Kategori Circuit Breaker

- **Metode yang membuat panggilan REST**—Hal ini memungkinkan untuk gagal dikarenakan ketidakadaan remote service atau mengembalikan respon HTTP 500.
- **Metode yang melakukan query database**—Hal ini memungkinka untuk gagal jika, untuk suatu alasan database menjadi tidak responsif atau perubahan skema yang merusak aplikasi.
- **Metode yang berpotensi lambat**—Hal-hal ini belum tentu gagal, tetapi dipertimbangkan tidak baik jika terlalu lama melakukan pekerjaan

Netflix Hyxtrix

- Netflix Hystrix adalah sebuah implementasi dari pola Circuit Breaker berbasis Java
- Mudahnya, Hystrix Circuit Breaker adalah implementasi dari aspek yang diterapkan ke metode yang memicu sebuah metode cadangan ketika terjadi kegagalan

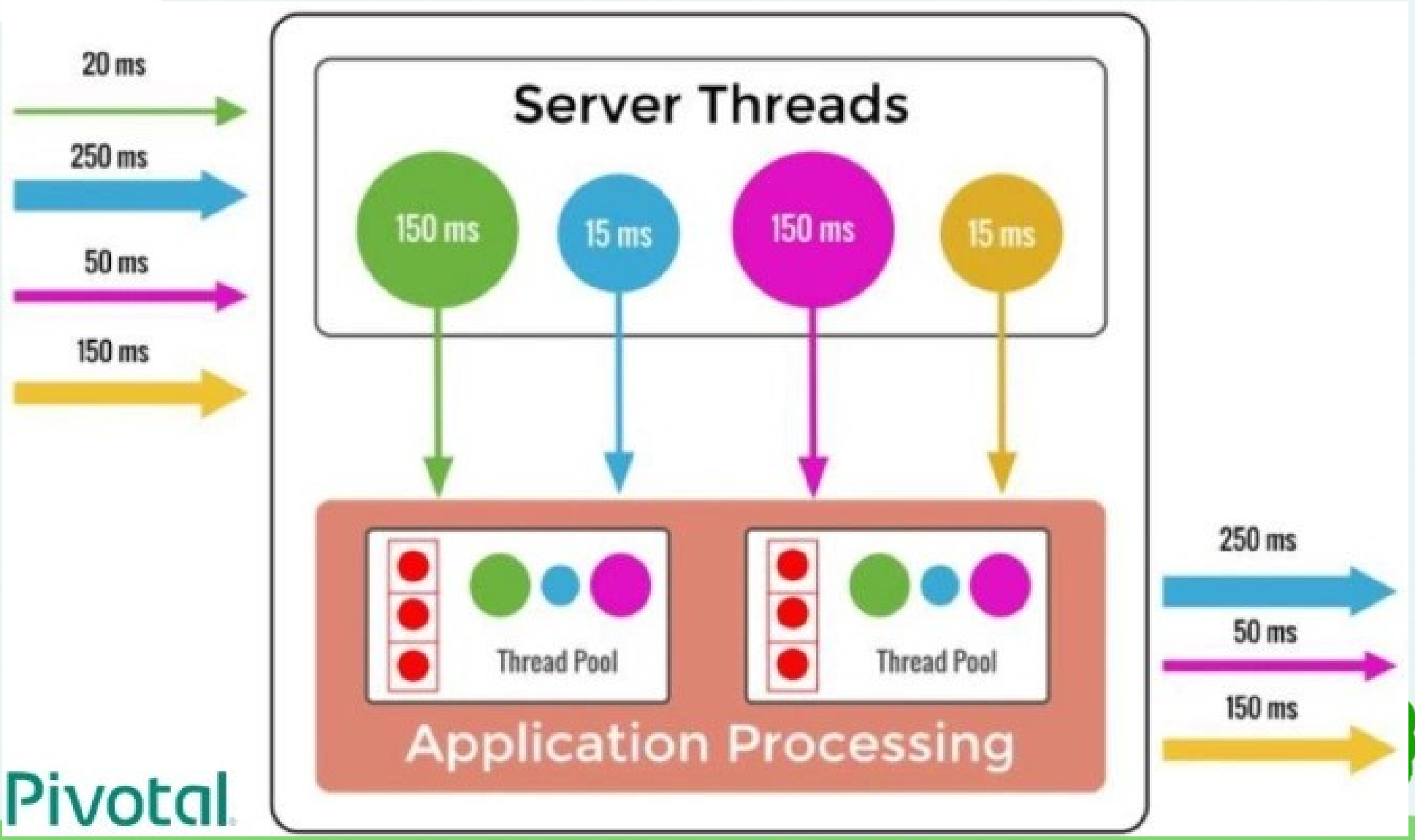
Mengatasi Latensi

- Latency adalah indikator sistem yang diukur dari request-respon
- Circuit Breaker juga bisa digunakan untuk menekan latensi dari timeout jika metodenya terlalu lama untuk berjalan.
- Secara Default semua metode yang teranotasi @HystrixCommand akan mengalami time out setelah 1 detik, dan kembali ke metode cadangan.

Lanjutan

- Penggunaan satu detik sebagai penentu merupakan hal yang masuk akal untuk seagala kebutuhan. Tetapi hal ini bisa diganti menjadi lebih ketat atau longgar.
- Konfigurasi perintah properti Hystrix bisa diatur melalui atribut command-properties.

Ilustrasi



Pivotal

Latency dan Bisnis

- Keberadaan latency dapat menentukan baik buruknya kegiatan bisnis khususnya yang bersifat online
- Latency menentukan sebuah website itu layak digunakan atau tidak, sehingga jika latency **cukup tinggi** bisa membuat pengguna enggan menggunakannya
- Google +500ms -20% Traffic
- Amazon +100ms -1% Sales

Mengatur Batas Circuit Breaker

- Secara default, jika metode proteksi circuit breaker di aktifkan lebih dari 20 kali dan lebih dari 50% aktifasi mengalami kegagalan lebih dari 10 detik, maka circuit akan di set ke rangkaian terbuka
- Semua panggilan berturut-turut akan dihandle oleh metode cadangan. Setelah 5 detik, circuit akan kembali ke mode setengah terbuka

Lanjutan

- Hal-hal berikut ini yang mempengaruhi circuit akan di set:
  - **circuitBreaker.requestVolumeThreshold** —Jumlah panggilan metode yang seharusnya dipanggil dalam satuan waktu
  - **circuitBreaker.errorThresholdPercentage** —Sebuah persentase dari metode yang gagal dipanggil dalam satu waktu
  - **metrics.rollingStats.timeInMilliseconds** —Sebuah waktu untuk request volume dan persentase error yang dipertimbangkan
  - **circuitBreaker.sleepWindowInMilliseconds** —Berapa lama rangkaian terbuka dibiarkan terbuka sebelum memasuki mode setengah terbuka (half-open) dan metode cadangan kembali dicoba

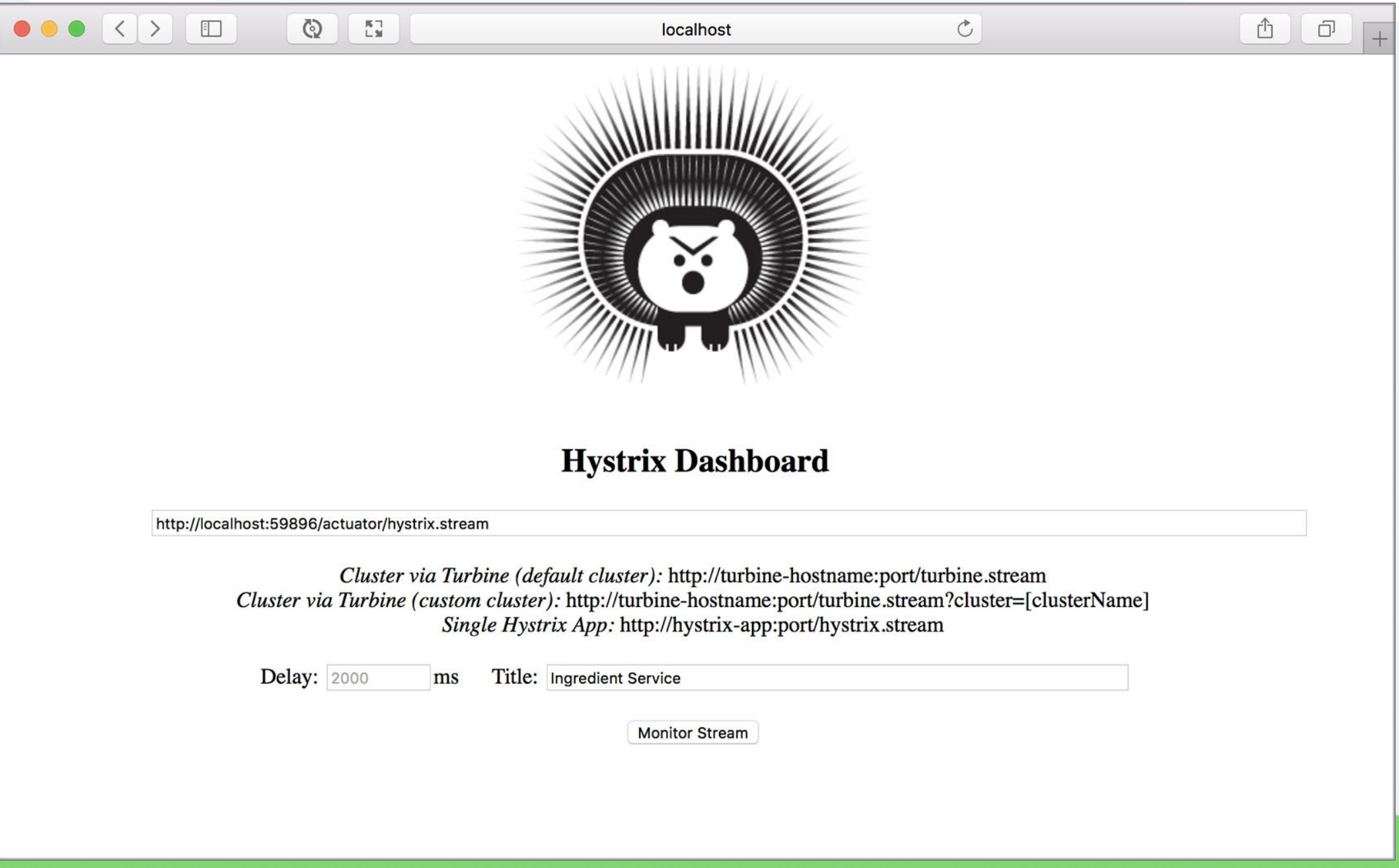
Monitor Kesalahan

- Setiap waktu metode proteksi circuit breaker dieksekusi, beberapa potong data akan dikoleksi mengenai eksekusi dan dipublikasikan melalui stream HTTP untuk kegunaan monitoring kesehatan

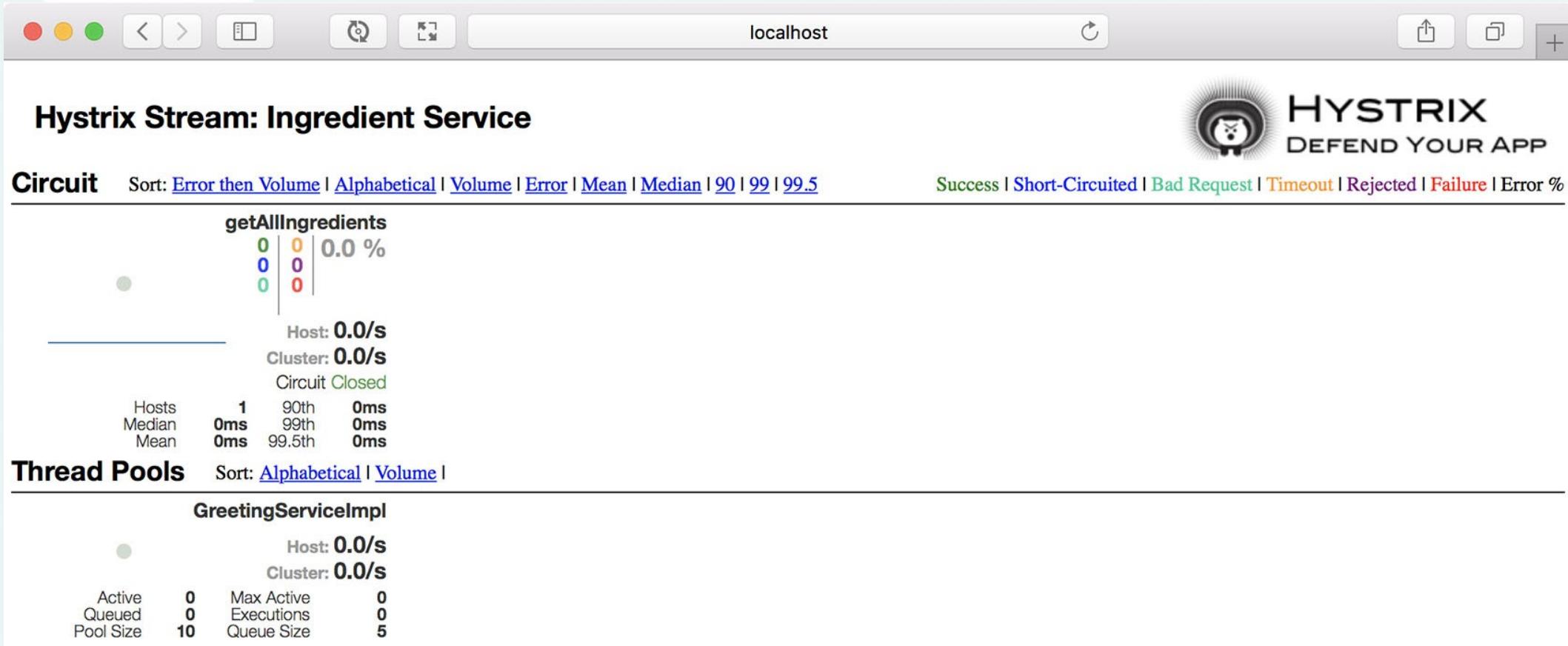
Data Monitor

- Hystrix akan melakukan pengoleksian data sebagai berikut:
  - Berapa kali metode dipanggil
  - Berapa kali metode berhasil dipanggil
  - Berapa kali metode cadangan dipanggil
  - Berapa kali metode mengalami time out

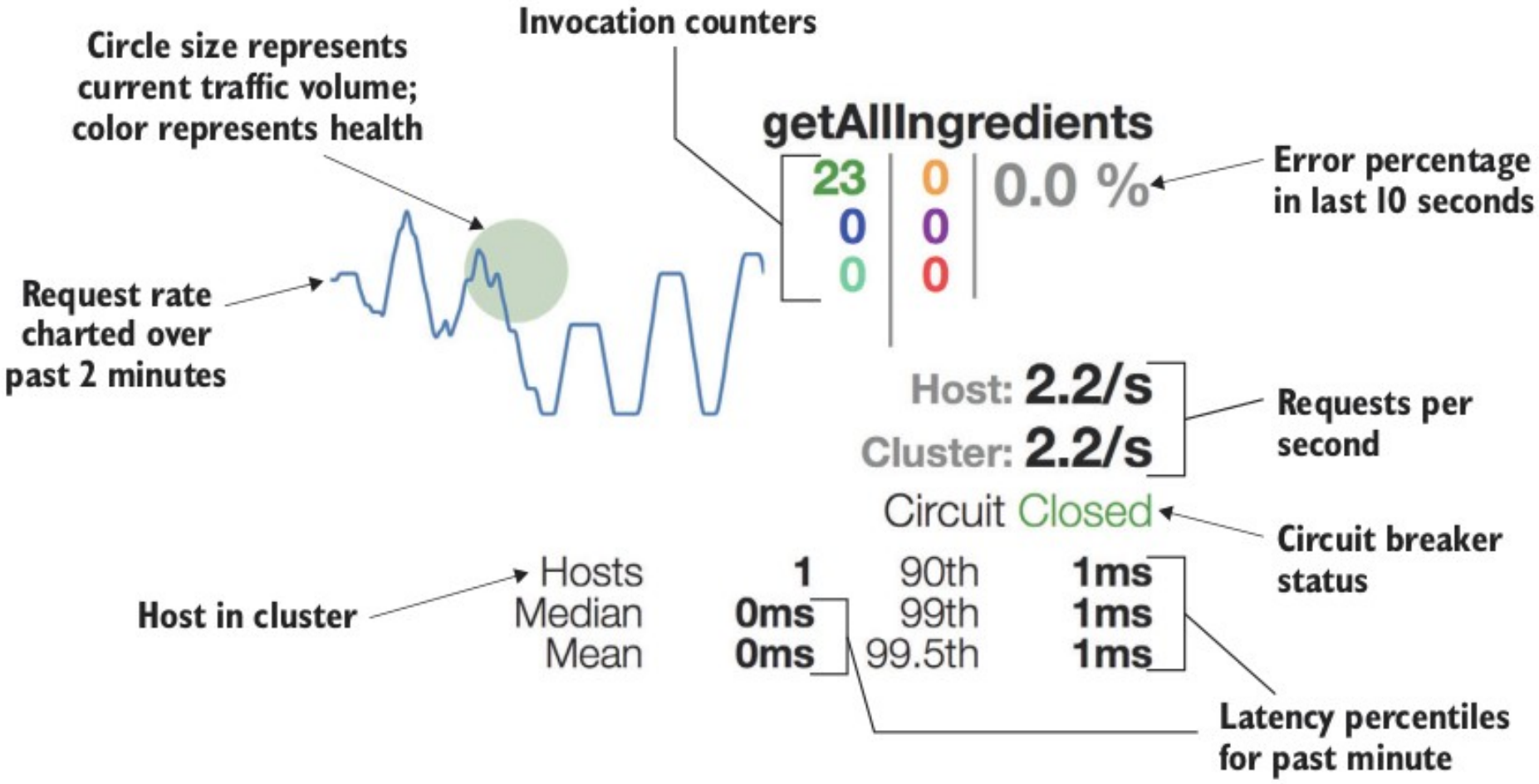
Tampilan


localhost
Hystrix Dashboard
http://localhost:59896/actuator/hystrix.stream
Cluster via Turbine (default cluster): http://turbine-hostname:port/turbine.stream
Cluster via Turbine (custom cluster): http://turbine-hostname:port/turbine.stream?cluster=[clusterName]
Single Hystrix App: http://hystrix-app:port/hystrix.stream
Delay: 2000 ms
Title: Ingredient Service
Monitor Stream

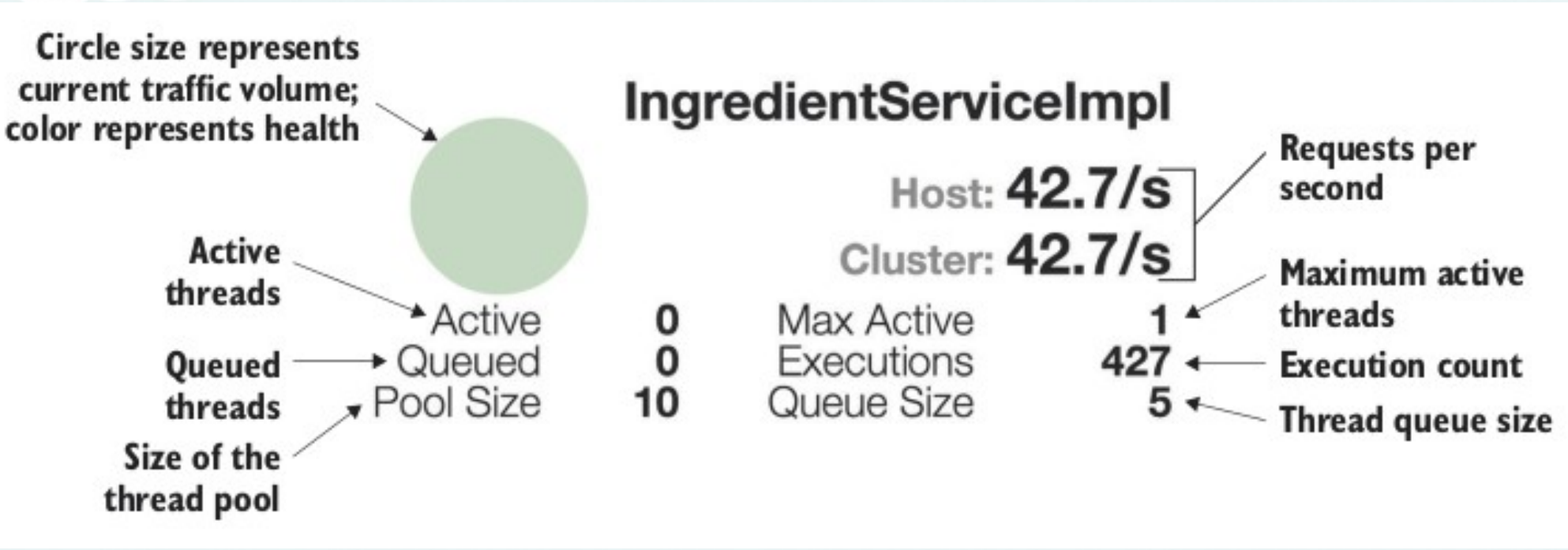
Tampilan


localhost
Hystrix Stream: Ingredient Service
HYSTRIX
DEFEND YOUR APP
Circuit
Sort: Error then Volume | Alphabetical | Volume | Error | Mean | Median | 90 | 99 | 99.5
Success | Short-Circuited | Bad Request | Timeout | Rejected | Failure | Error %
getAllIngredients
0 0 0.0 %
0 0
0 0
Host: 0.0/s
Cluster: 0.0/s
Circuit Closed
Hosts 1 90th 0ms
Median 0ms 99th 0ms
Mean 0ms 99.5th 0ms
Thread Pools
Sort: Alphabetical | Volume |
GreetingServiceImpl
Host: 0.0/s
Cluster: 0.0/s
Active 0 Max Active 0
Queued 0 Executions 0
Pool Size 10 Queue Size 5

Tampilan


Circle size represents current traffic volume; color represents health
Invocation counters
getAllIngredients
23
0
0
0
0
0
0.0 %
Error percentage in last 10 seconds
Request rate charted over past 2 minutes
Host: 2.2/s
Cluster: 2.2/s
Requests per second
Circuit Closed
Circuit breaker status
Host in cluster
Hosts 1
Median 0ms
Mean 0ms
90th 1ms
99th 1ms
99.5th 1ms
Latency percentiles for past minute

Tampilan

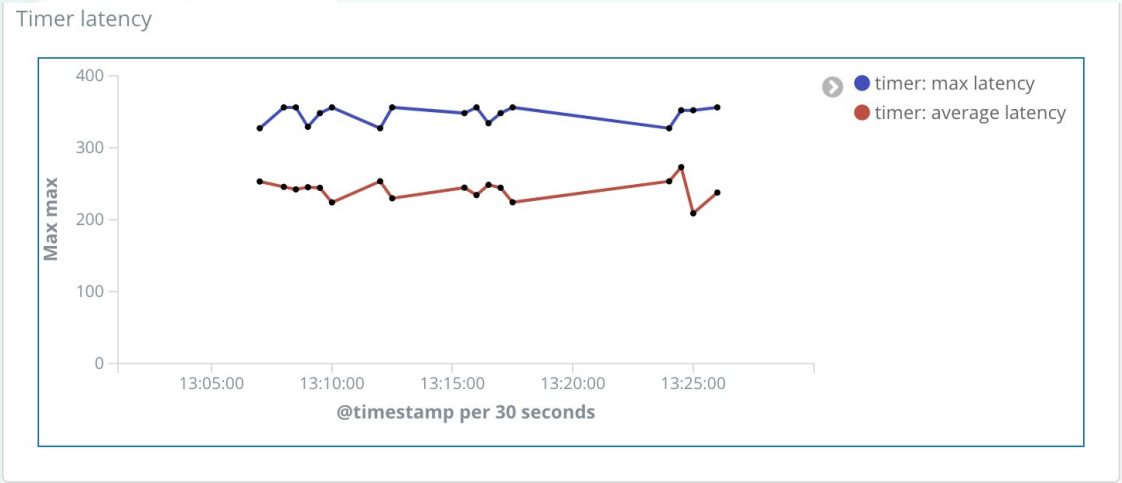

Circle size represents current traffic volume; color represents health
IngredientServiceImpl
Host: 42.7/s
Cluster: 42.7/s
Requests per second
Active threads
Active
0
Max Active
1
Maximum active threads
Queued threads
Queued
0
Executions
427
Execution count
Size of the thread pool
Pool Size
10
Queue Size
5
Thread queue size

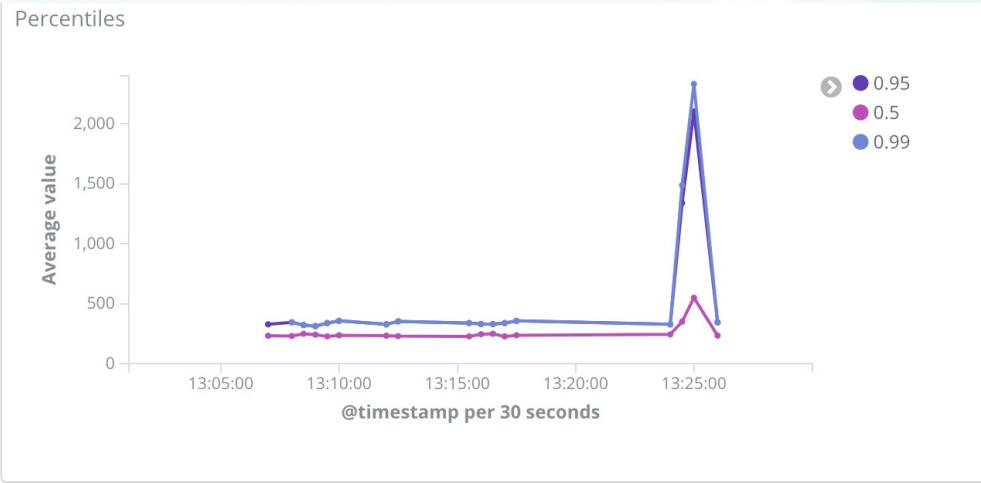
Spring Boot Micrometer

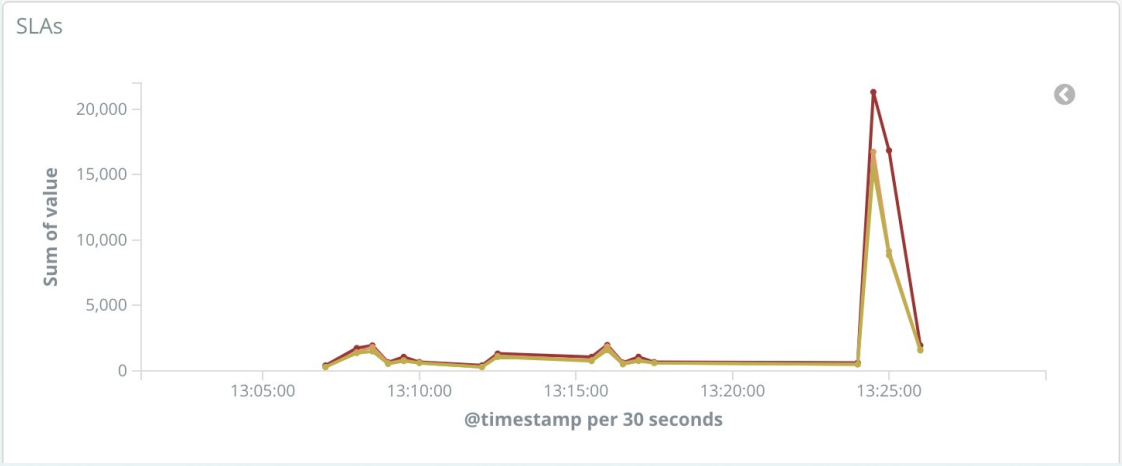
- Sistem monitor oleh Spring Boot yang berguna untuk melakukan monitoring
- Mirip dengan SLF4J tetapi dalam bentuk metrik data

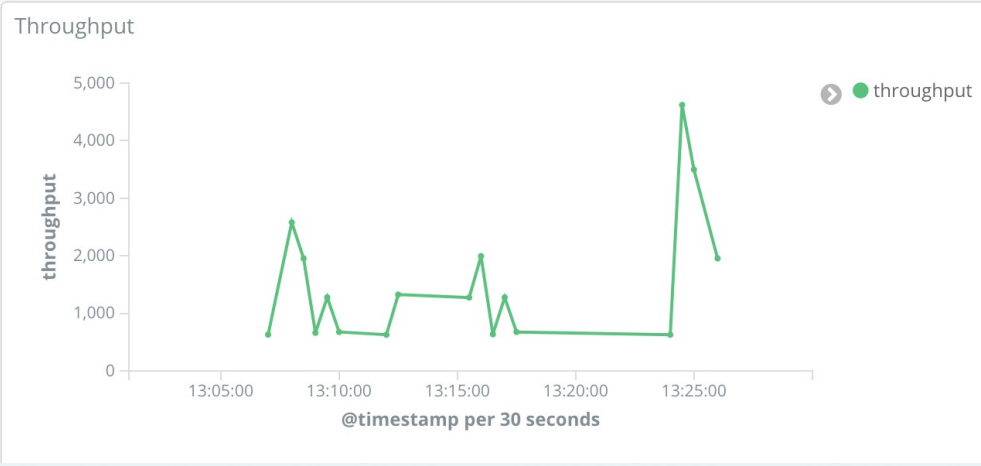
- Micrometer menambahkan meter primitif yang lebih baik dari versi sebelumnya. Contohnya variable Timer dapat memproduksi semua diagram berbasis Time/Waktu

Lanjutan


Timer latency
timer: max latency
timer: average latency
Max max
400
300
200
100
0
13:05:00
13:10:00
13:15:00
13:20:00
13:25:00
@timestamp per 30 seconds


Percentiles
0.95
0.5
0.99
Average value
2,000
1,500
1,000
500
0
13:05:00
13:10:00
13:15:00
13:20:00
13:25:00
@timestamp per 30 seconds


SLAs
Sum of value
20,000
15,000
10,000
5,000
0
13:05:00
13:10:00
13:15:00
13:20:00
13:25:00
@timestamp per 30 seconds


Throughput
throughput
throughput
5,000
4,000
3,000
2,000
1,000
0
13:05:00
13:10:00
13:15:00
13:20:00
13:25:00
@timestamp per 30 seconds

Pencegahan Latency Tinggi

- Memilih bahasa yang tepat
  - Sebaiknya memilih bahasa dengan memory model yang kuat, sehingga terhindari dari latency tinggi
  - Sebagai contoh, Java, C++, Scala

Lanjutan

- Simpan semua di memory
  - I/O akan mematikan latency, jadi sebaiknya semua data di load di memory
  - Maksudnya memanajemen data memory internal dan log di dalam memory supaya bisa dibangung ulang setelah restart

Lanjutan

- Pastikan data dan proses teralokasikan
  - Jaringan lebih cepat dari putaran disk, sehingga idealnya data-data sudah teralokasikan di dalam disk
- Pastikan sistem tidak sibuk
  - Latency rendah memerlukan sumber daya yang selalu tersedia untuk berjaga-jaga dari burst

Lanjutan

- Cache
  - Dengan adanya optimalisasi diberbagai tempat, akses memory menjadi bottle neck
  - Gunakanlah algoritma cache yang bekerja secara rekursif memecah data sehingga muat di cache
- Paralelisme
  - Segala proses yang harus dikerjakan secara paralel sebaiknya dikerjakan secara paralel juga