Pivotal

Spring Cloud Netflix Hystrix - Fallback Handling

Problem

- What happens if I cannot reasonably protect against dependencies in my code?
  - Unhandled exceptions
  - Tie up/deplete threads in my application
  - Potentially run for a long time, where I cannot gracefully handle a timeout.
  - I need fine grained monitoring for either of above failures

What does it do?

- Provides graceful means of handling failure in an application.
  - In protected code, divert call execution path to a *Fallback* method upon failures.
- Failure modes protected:
  - Thread pool depletion
  - Hard Timeout
  - Unhandled Runtime exceptions

How does it work?

- Hystrix client runs in-process to the application being protected
- Hystrix Command uses AOP (and associated Spring Dynamic Proxies) to wrap protected code.
- Hystrix Proxies generate keyed thread pool used to run wrapped (protected) method, unless using Semaphores (covered later in this course).
- Hystrix Proxies will divert call execution path to a *Fallback* method upon failures.

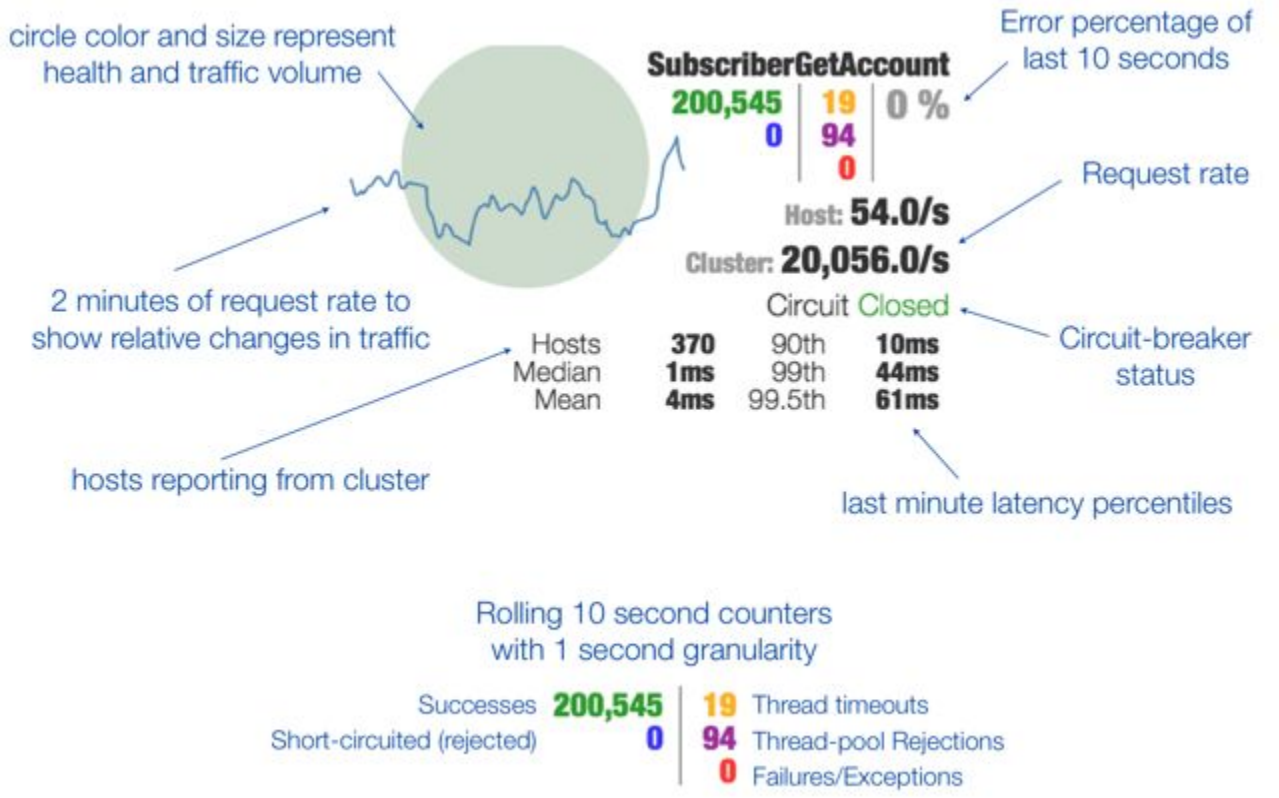
See following for more info:

- https://github.com/Netflix/Hystrix/wiki/How-it-Works#threads--thread-pools

Hystrix Dashboard

- A standalone application providing visualization of protected command state
- Calling application emits metrics via endpoint `/actuator/hystrix.stream`
- Dashboard application consumes stream and renders metrics visualization for each protected command.

Hystrix Monitoring



Trade-Offs

- Cannot unit test it
- Protected code must be thread-safe
- Operational and Tuning Complexity
- Requires close production monitoring and tuning
- Use Hystrix sparingly, only where needed
- Require careful design considerations for write operations.

You will see more about specific trade-offs in later sections of the course.