Hands-On: Hystrix
Best practices & pitfalls
Hystrix...What?

- built, heavily tested & used in production by Netflix
- Java library, implementation of resilience patterns
- **Goals**: fault tolerant/robust self-healing applications, zero downtime
Which cases
 › Distributed system
 › External dependencies
 › Limited resources
 › High Availability
 › Avoid incidents
Availability

> 99,9% (three nines) = 9 hour downtime per year
> 5 dependencies with 99,9% availability = 99,9%\(^5\) = 99,5% = 9 days downtime per year
How

- Decoupling by asynchronous execution
- Safety first: bulkheads, fail fast
- Circuit Breaker
- Fallback, graceful degradation
Where

- Close to the action
- Isolate each dependency separately
- 2+ external calls per single client API call: wrap all dependencies additionally
Example: Synchronous Mail delivery

```java
@EnableHystrix
@SpringBootApplication
public class ShopApplication {}  

[RegisterService]
@Transactional
public void register(UserDto userDto) {
    User user = userRepo.save(map(userDto));
    mailService.sendRegisterSuccessMail(map(user));
}

[MailService]
@HystrixCommand
public void sendRegisterSuccessMail(UserDto user) {
    mailGateway.sendMail('register_success', user);
}
```

3rd Party Libs: Spring Boot, Spring Cloud Netflix (auto setup- & shutdown-routines)
Example: Delayed Order

```java
@HystrixCommand(fallbackMethod = "placeOrderFallback")
public OrderResult placeOrder(Order order) {
    long orderId = orderService.placeOrder(order);
    order.setOrderId(orderId);
    order.setStatus('RUNNING');
    return OrderResult.ok(orderId);
}

class OrderResult {
    public static final OrderResult OK = new OrderResult();
    public static final OrderResult OK_DELAYED = new OrderResult();
    public static final OrderResult NOT_FOUND = new OrderResult();
    public static final OrderResult BAD_REQUEST = new OrderResult();
    public static final OrderResult SERVER_ERROR = new OrderResult();
    ...
}
```

```java
public OrderResult placeOrderFallback(Order order) {
    // place order async (e.g. cron job)
    return OrderResult.OK_DELAYED;
}
```
Example: Plain Hystrix

```java
public class OrderCommand extends HystrixCommand {

    public OrderCommand(OrderService orderService, Order order) {
        super(...config...);
        this.orderService = orderService;
        this.order = order;
    }

    @Override
    protected void run() {
        long orderId = orderService.placeOrder(order);
        return OrderResult.ok(orderId);
    }
}

new OrderCommand(orderService, order).execute();
```
Hystrix by Annotations: Javanica

- Smaller configuration effort
- Less boiler plate code
- Automatic unwrapping of HystrixBadRequestExceptions
- Aspect != Aspect
  - Possibly skip other advices
- **buggy: v1.5.4** faulty unwrapping of HystrixRuntimeExceptions
Semaphores & thread-pools

- Semaphore
  - concurrency & error control
  - remains in the caller thread
  - timeout != timeout
- Thread-pool
  - Hystrix (& Netflix internal) standard
  - real decoupling
  - async execution flow
JAVA THREADS ARE INTERRUPTIBLE
SET TIMEOUT! ALWAYS!
Timeouts

- HTTP connection/socket timeouts
- connection pools
- SQL driver
- Hystrix timeouts & limits
- timeout values: It depends!
Protecting threads

- Thread interruption is a desire, not a constraint
- Use non-blocking API (NIO Channel, Netty, ...)
- Handle thread interruption appropriately
  - Check: `Thread.currentThread().isInterrupted()`
  - Restore state after catching `InterruptedException`: `Thread.currentThread().interrupt();`
Decoupling not for free

- minimal overhead, **but**
- ThreadLocal powered functionality missing
  - log context (MDC, NDC)
  - DB transaction status
  - security context
  - thread bound DI scopes (request, session, thread)
Solution: Custom Strategy

```java
public class CustomStrategy extends HystrixConcurrencyStrategy {
    @Override
    public <T> Callable<T> wrapCallable(final Callable<T> callable) {
        final Map context = MDC.getContext();
        final RequestAttributes attr = RequestContextHolder.get();
        return super.wrapCallable(new Callable<T>() {
            @Override public T call() throws Exception {
                try {
                    MDC.replaceAll(context);
                    RequestContextHolder.set(attr);
                    return callable.call();
                } finally {
                    RequestContextHolder.resetRequestAttributes();
                    MDC.clear();
                }
            }
        });
    }

    HystrixPlugins.getInstance()
        .registerConcurrencyStrategy(new CustomStrategy());
}
```
Hystrix Dashboard

> Standalone/self-contained version (https://git.io/vPH8g)
> Consumes Hystrix metrics stream from each app
> Aggregated metrics stream with Hystrix Turbine
Demo time!

> API Gateway (JHipster)
> Angular JS / Bootstrap
> Spring Boot
> Hystrix
> 3 simple microservices
> Monitoring
> Hystrix dashboard
> Kibana dashboard (ELK)
More Hystrix

- HystrixObservableCommand: Wrapping non-blocking code (semaphore only)
- Hystrix command collapsing
- Hystrix request cache
Vielen Dank!

Gerrit Brehmer
Software Architekt/Senior Developer

inovex GmbH
Office Karlsruhe
Ludwig-Erhard-Allee 6
76131 Karlsruhe

Mobil: 01733181007
Mail: gerrit.brehmer@inovex.de