Creating Self-healing Services with Circuit Breaker

Dustin Schultz
SOFTWARE ENGINEER

@schultzdustin http://dustin.schultz.io/ dustin@schultz.io

Outline

Failures in a distributed system

- Cascading failures
- Circuit breaker pattern

Netflix Hystrix project

- `@EnableCircuitBreaker`
- `@HystrixCommand`

Hystrix Dashboard

- `@EnableHystrixDashboard`
- Turbine to aggregate Hystrix streams
  - `@EnableTurbine`

In a Distributed System
one thing is absolutely certain

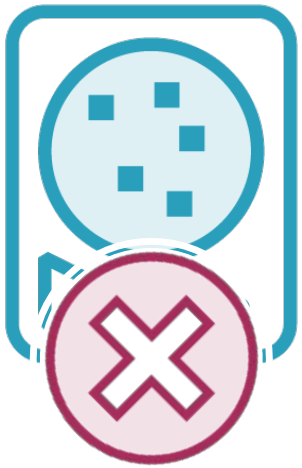
...

FAILURE

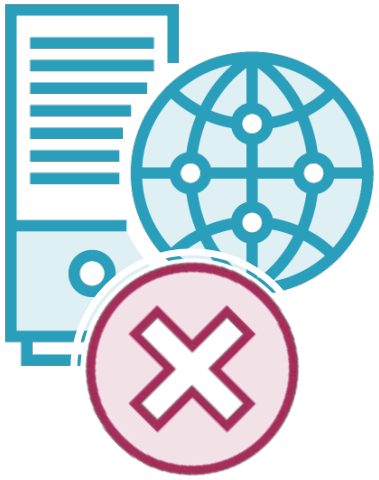
IS INEVITABLE

But why?

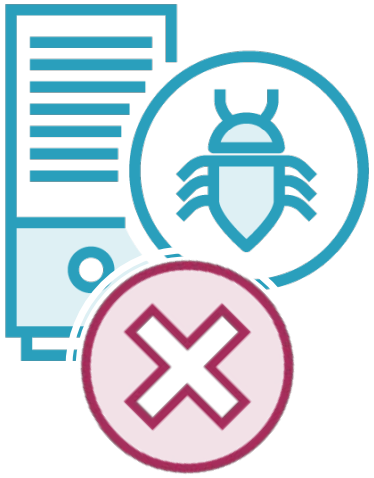
A Few Areas That Might Fail



Hardware fails



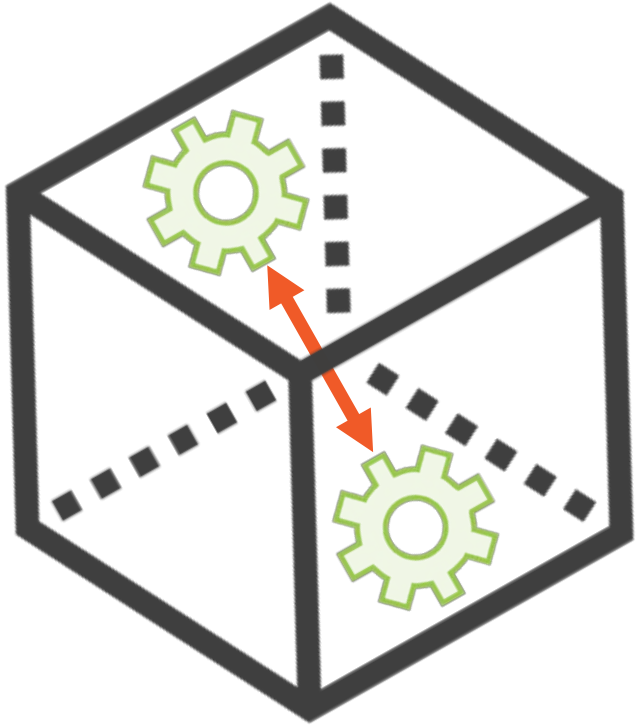
Networks fail



Software fails

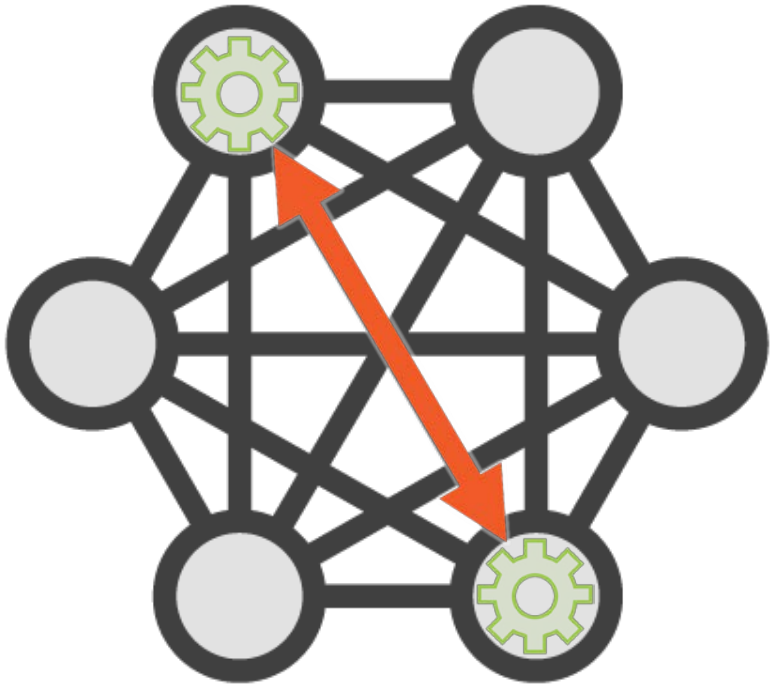
That chance of
failure becomes multiplied
in a distributed system

Process Communication Is Also More Likely to Fail



Within the process

to ...



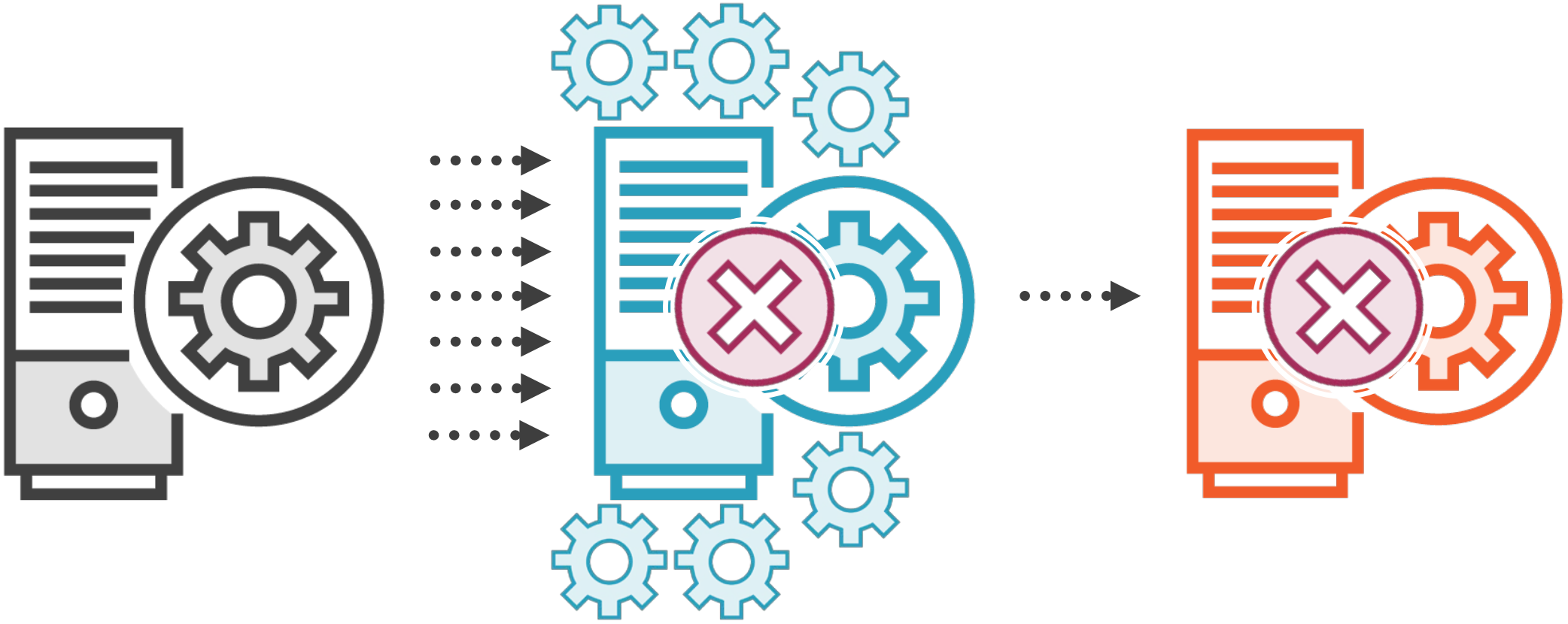
Across the network

Cascading Failure

“ … a failure in a system of interconnected parts in which the failure of a part can trigger the failure of successive parts.”

- *Wikipedia*

Bad Side Effects: Cascading Failures



Service A

Service B

Service C

Multiple issues at play

- Fault tolerance problem
- Resource overloading problem

So, what can we do?
How can we *solve* this?

Learn to *embrace failure*

- Tolerate failures
- Gracefully degrade

Limit resources consumed

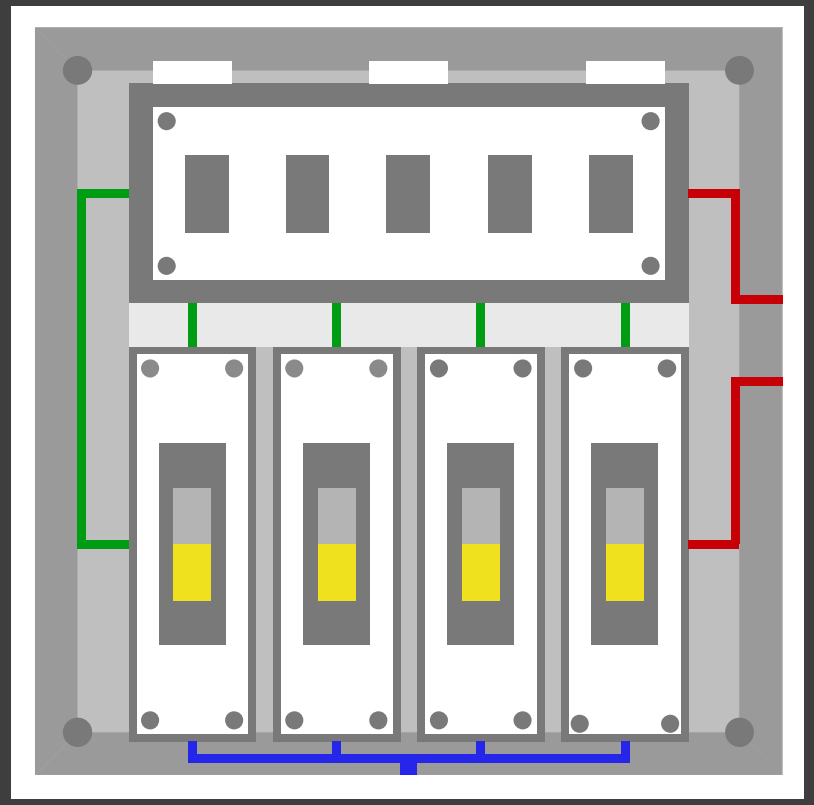
- Constrain usage

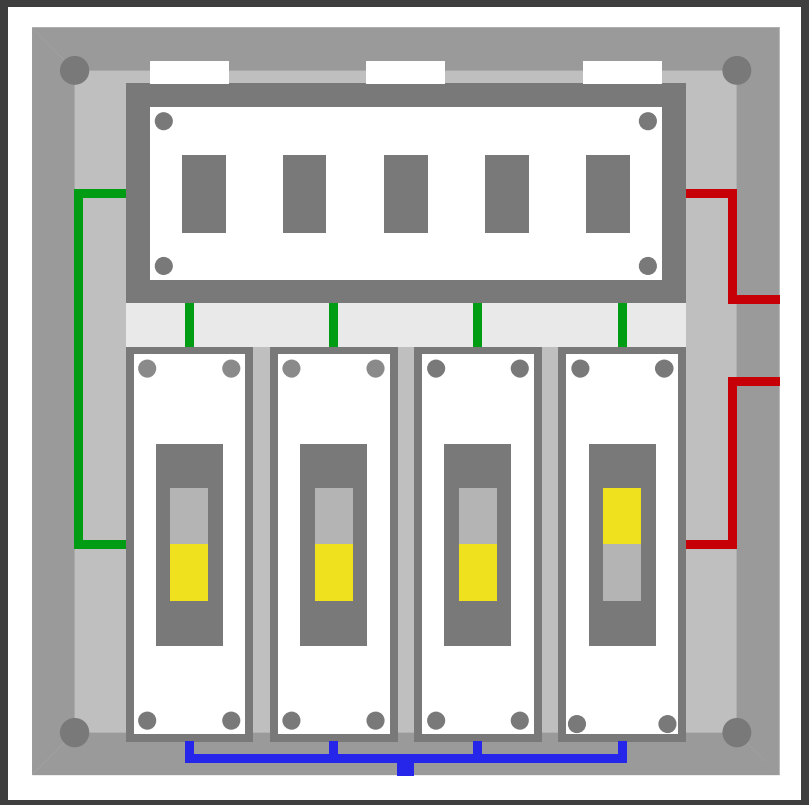
Circuit Breaker Pattern

"... a design pattern in modern software development used to detect failures and encapsulates logic of preventing a failure to reoccur constantly ..."

- *Wikipedia*

Circuit Breaker





Circuit *closed*

Circuit *open*

Fault Tolerance with Netflix Hystrix and Spring Cloud

Netflix Hystrix

Hystrix is a latency and **fault tolerance** library designed to stop **cascading failure** and enable resilience in complex distributed systems **where failure is inevitable**.

-Netflix Hystrix Project page

Implements the circuit breaker pattern

- Wraps calls and watches for failures
  - 10 sec rolling window
  - 20 request volume
  - >= 50% error rate
- Waits & tries a single request after 5 sec
- Fallbacks

Protects services from being overloaded

- Thread pools, semaphores, & cascading failures

Using Spring Cloud & Netflix Hystrix

pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Camden.SR2</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Using Spring Cloud & Netflix Hystrix

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-actuator</artifactId>
</dependency>
```

Using Spring Cloud & Netflix Hystrix

Application.java

```
@SpringBootApplication
@EnableCircuitBreaker
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Using the `@HystrixCommand` Annotation

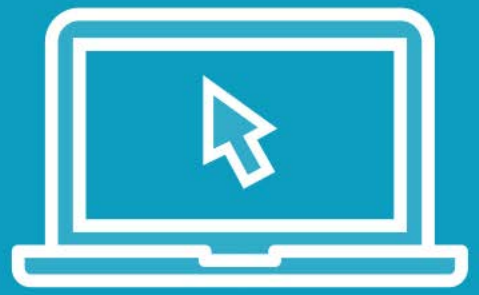
Service.java

```
@Component
public class Service {

    @HystrixCommand(fallbackMethod = "somethingElse")
    public void doSomething() {
        ...
    }

    public void somethingElse() {
        ...
    }
}
```

Be careful with Hystrix timeouts

- Ensure timeouts encompass caller timeouts plus any retries
- Default: 1000ms
- `hystrix.command.default.execution.isolation.thread.timeoutInMilliseconds=<timeout_ms>`

Demo



Using the `@EnableCircuitBreaker` and `@HystrixCommand` annotations

https://github.com/dustinschultz/scf-discovery-server

Monitor Hystrix Metrics in Real Time with the Hystrix Dashboard

What Is the Hystrix Dashboard?

Tracks metrics such as

Circuit state

Error rate

Traffic volume

Successful requests

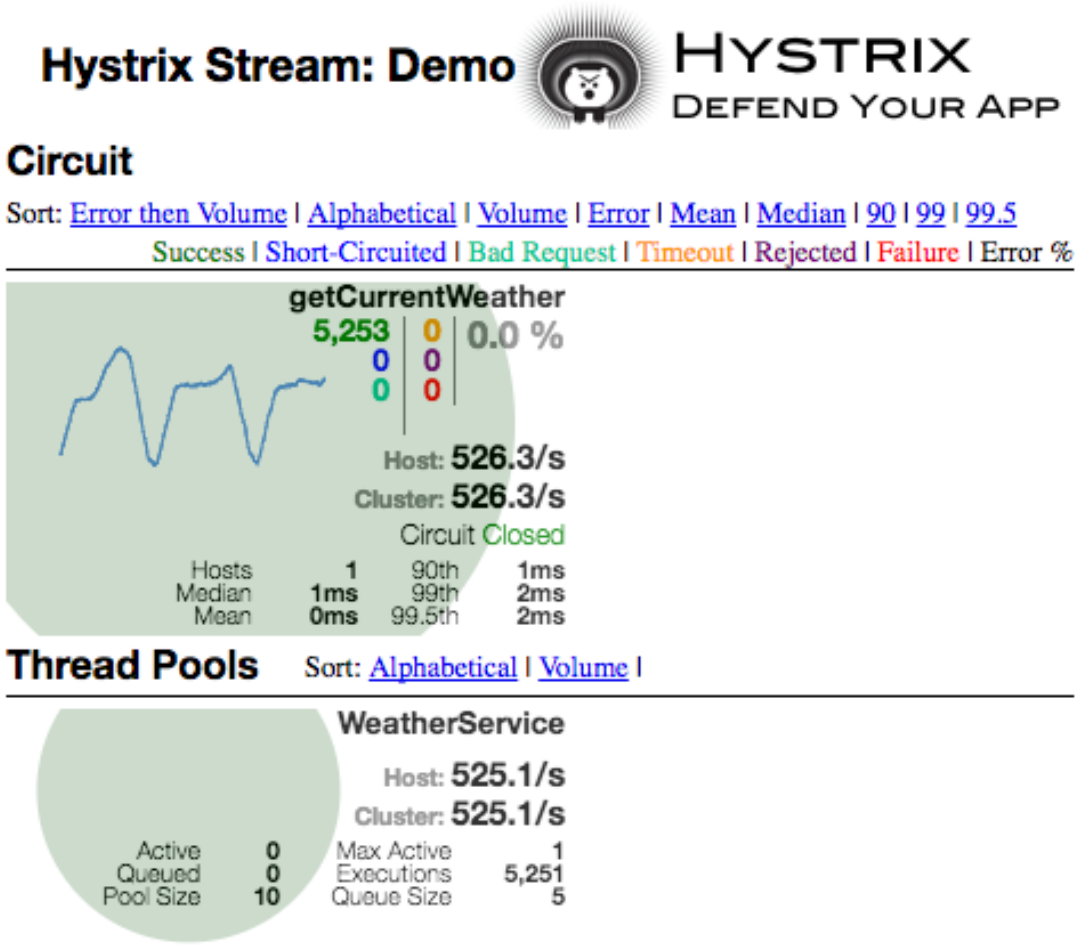
Rejected requests

Timeouts

Latency percentiles

Monitor protected calls

Single server or cluster



Using Spring Cloud & Netflix Hystrix Dashboard

pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Camden.SR2</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Using Spring Cloud & Netflix Hystrix Dashboard

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix-dashboard</artifactId>
</dependency>
```

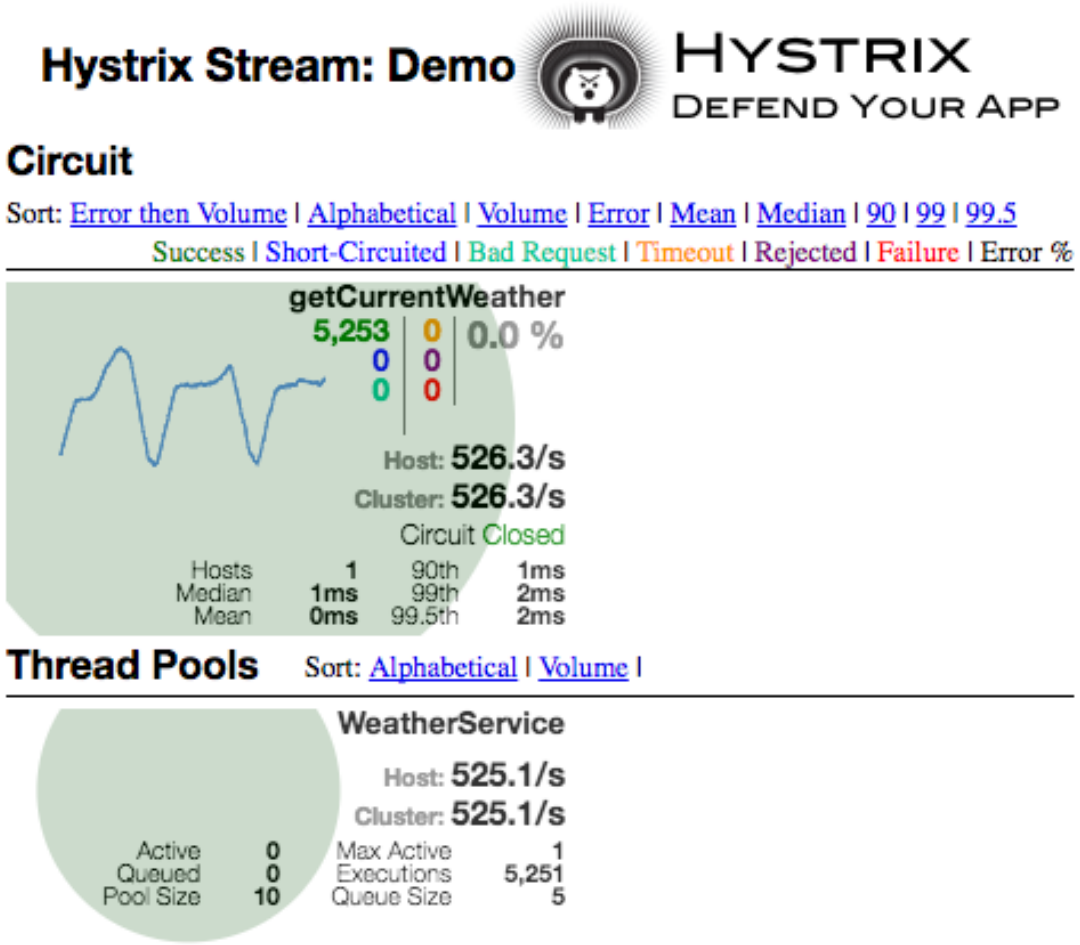
Using Spring Cloud & Netflix Hystrix Dashboard

Application.java

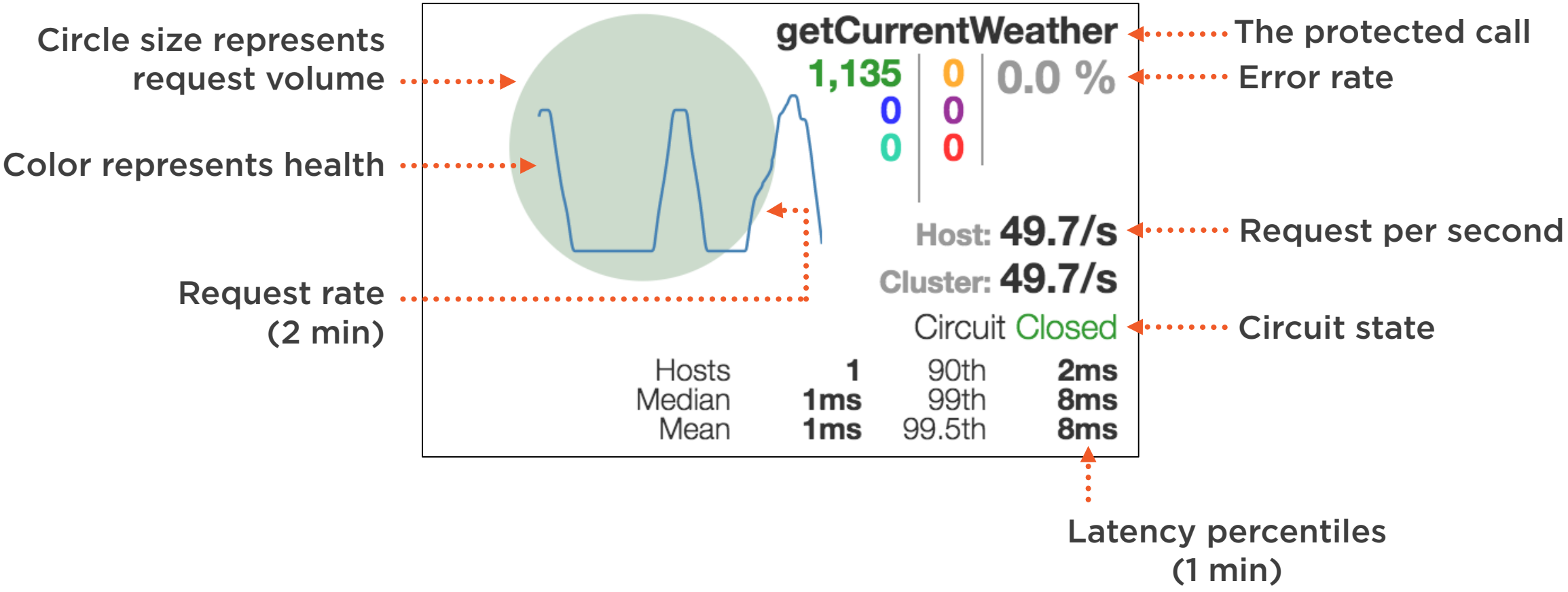
```java
@SpringBootApplication
@EnableHystrixDashboard
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Understanding the dashboard

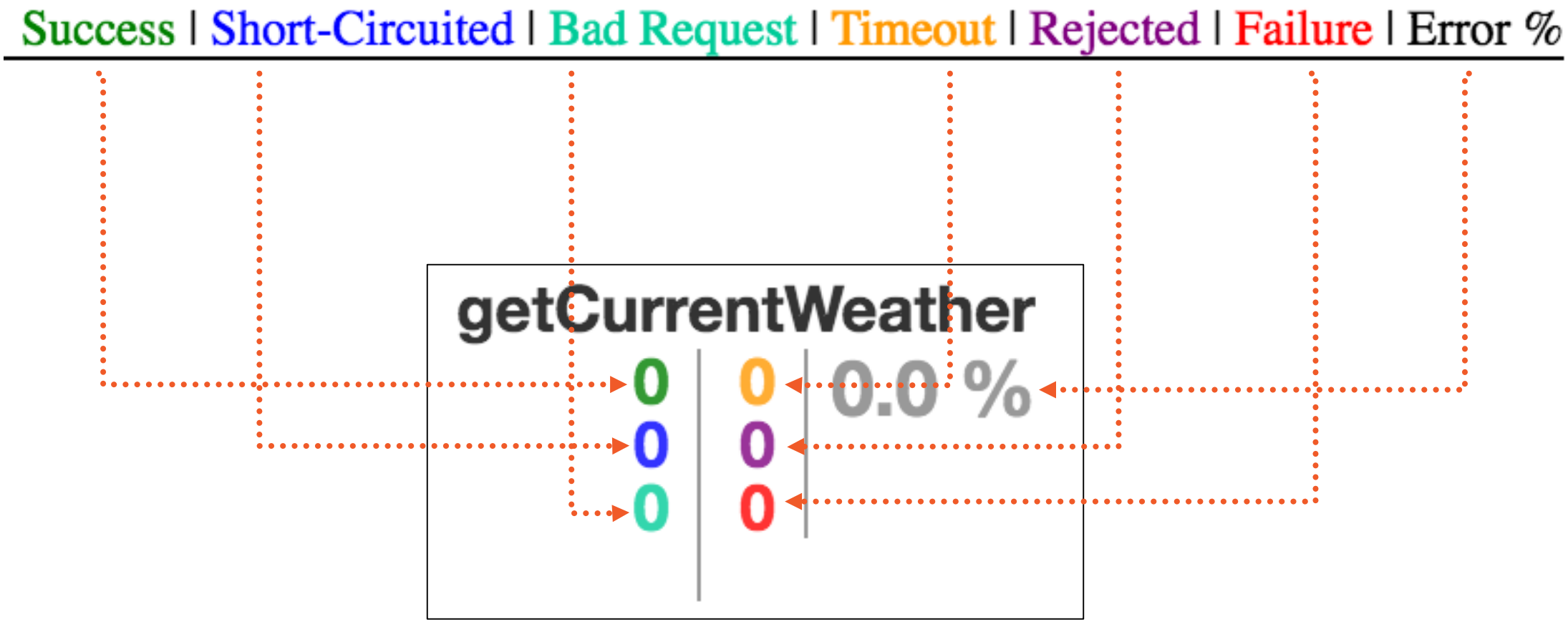
It contains a LOT of information in a little amount of space



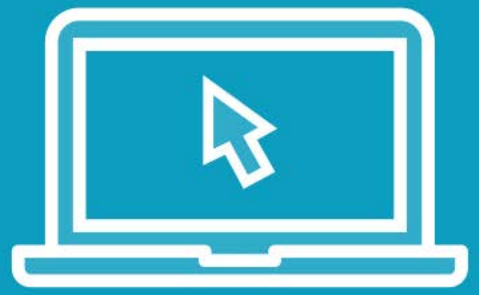
How to Read the Hystrix Dashboard



How to Read the Hystrix Dashboard



Demo



Using `@EnableHystrixDashboard`

Aggregating Hystrix Streams with Turbine

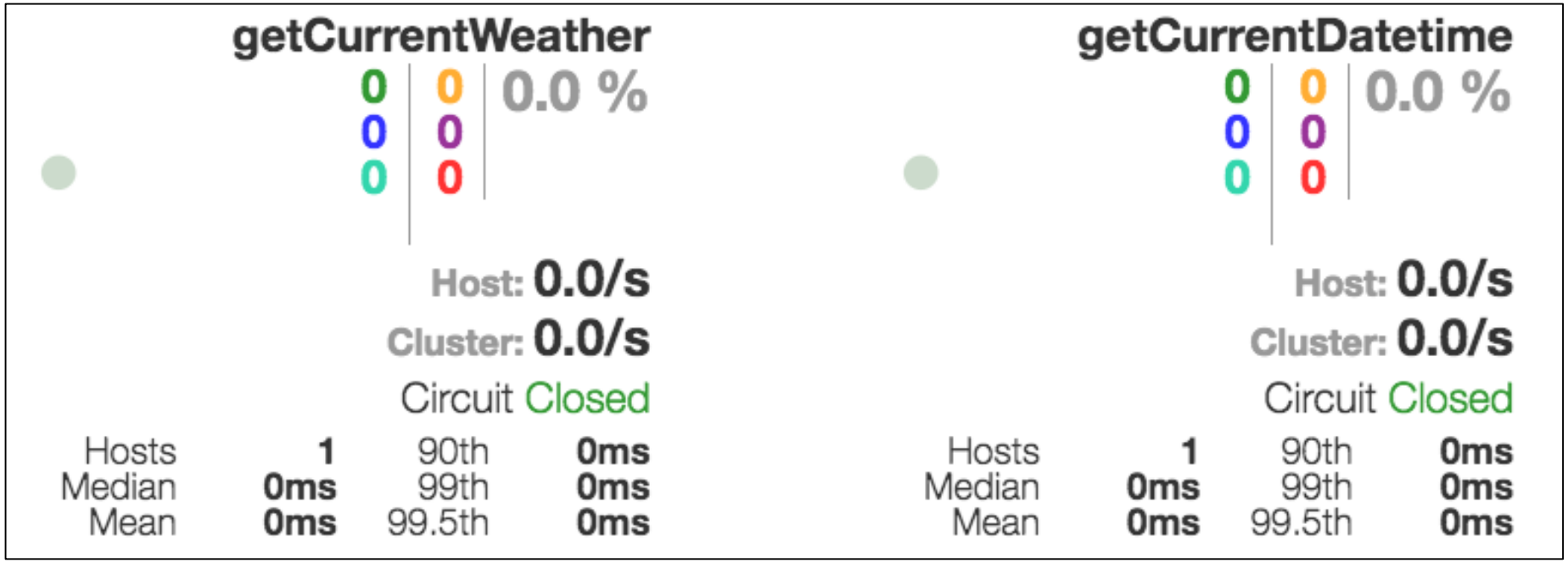
Viewing multiple Hystrix metrics, **all at different URLs**, could make you very grumpy!

Netflix Turbine

"Turbine is a tool for **aggregating streams** of Server-Sent Event (SSE) JSON data **into a single stream**..."

- Netflix Turbine Project Page

Multiple Hystrix Streams in One View on the Hystrix Dashboard



Stream located at `localhost:8080`

Stream located at `localhost:8181`

Using Spring Cloud & Netflix Turbine

pom.xml

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>Camden.SR2</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Using Spring Cloud & Netflix Turbine

pom.xml

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-turbine</artifactId>
</dependency>
```

Using Spring Cloud & Netflix Turbine

Application.java

```
@SpringBootApplication
@EnableTurbine
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Using Spring Cloud & Netflix Turbine

application.properties

```
turbine.app-config=<list_of_service_ids>
turbine.cluster-name-expression='default'
```

OR

application.yml

```
turbine:
  appConfig: <list_of_service_ids>
  clusterNameExpression: "'default'"
```

* In addition to the standard spring application name and discovery server location properties

Netflix Turbine and the Hystrix Dashboard

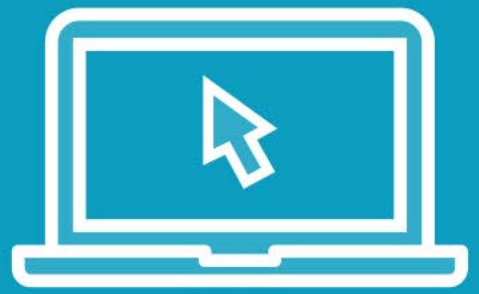
Hystrix Dashboard

http://hostname:port/turbine.stream

https://github.com/dustinschultz/scf-hystrix-datetime-service

https://github.com/dustinschultz/scf-hystrix-datetime-app

Demo



Using `@EnableTurbine`

Summary

Fault tolerance is a requirement

Netflix Hystrix

- Circuit breaker pattern
- `@HystrixCommand` & `@EnableCircuitBreaker`

Netflix Hystrix Dashboard & Turbine

- Monitor one or several streams