Course 3 – Sprint 8: Handle the failure of Microservices by implementing Circuit Breaker pattern Using Hystrix

In the world of Microservices it is common that some services fail and this failure paves way for cascading failures and other microservices too will have impact. In order to avoid it we make use of a fault-tolerance mechanism, Circuit Breaker.

What is a Fallback Mechanism in Microservices?

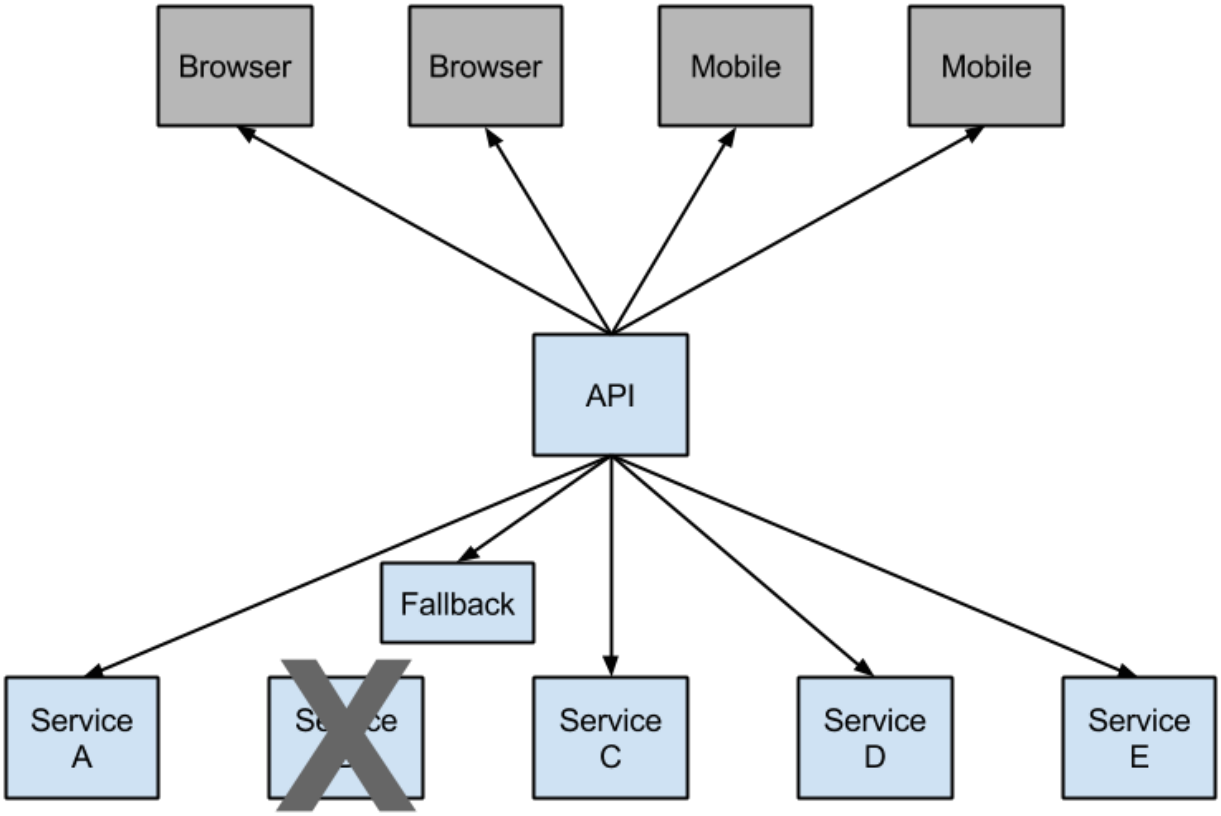
Fallback provides an alternative solution during a service request failure. When the circuit breaker trips and the circuit is open, a fallback logic can be started instead.

The fallback logic typically does little or no processing and return value. Fallback logic must have a little chance of failing, because it is running as a result of a failure to begin with.

Circuit Breaker:

The circuit breaker is fault-tolerance technique that monitors and detects when a service is behaving abnormally.

A service failure in the lower level of services can cause cascading failure all the way up to the user. When calls to a particular service exceed circuitBreaker.requestVolumeThreshold (default: 20 requests) and the failure percentage is greater than circuitBreaker.errorThresholdPercentage (default: >50%) in a rolling window defined by metrics.rollingStats.timeInMilliseconds (default: 10 seconds), the circuit opens and



the call is not made. In cases of error and an open circuit, a fallback can be provided by the developer.

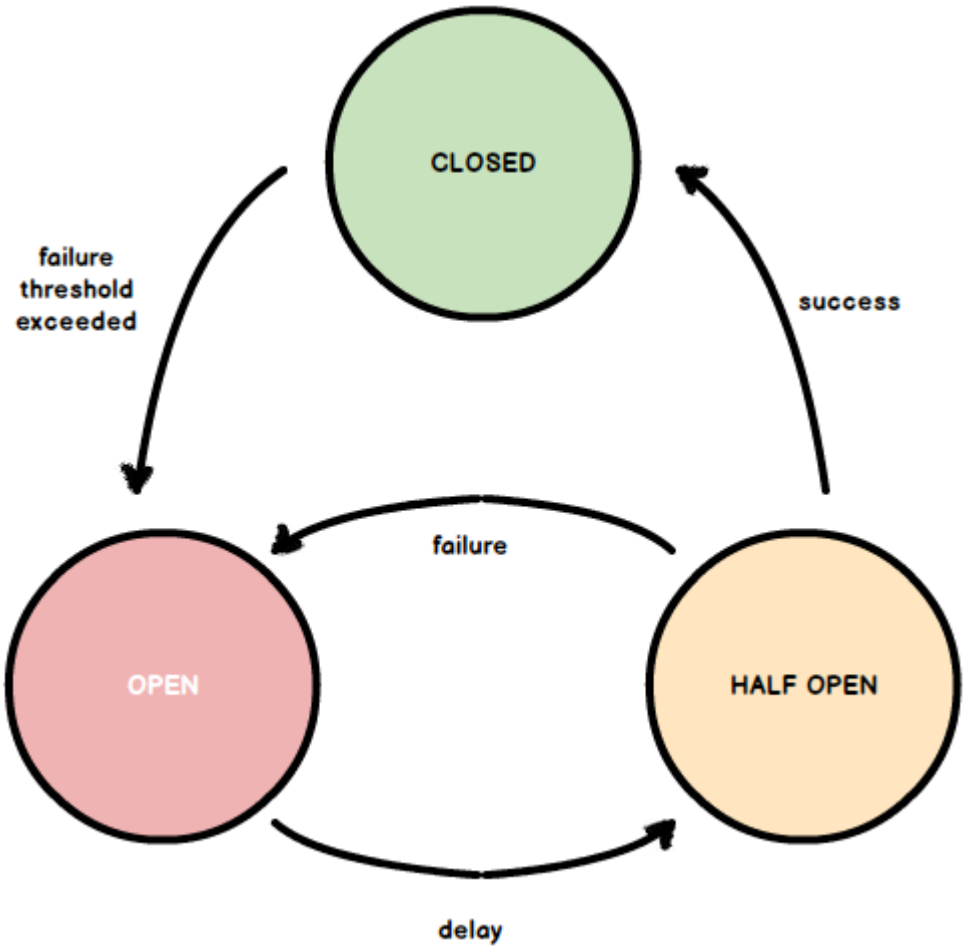
Having an open circuit stops cascading failures and allows overwhelmed or failing services time to recover. The fallback can be another Hystrix protected call, static data, or a sensible empty value. Fallbacks may be chained so that the first fallback makes some other business call, which in turn falls back to static data.

Different states of Circuit Breaker

Circuit Breaker can be in any one of the following states,

1. Open
2. Closed

3. Half-Open



Courtesy: https://medium.com/@narengowda/what-is-circuitbreaking-in-microservices-2053f4f66882

Open State:

If any request is raised and gets failed because of any exception in the method call or service is down, then it is said to be open state. Hence, this state represents that service is unavailable or faulty and error rate is beyond the threshold.

Closed State:

Any request raised gets executed normally without any problem then it is said to be closed state. So, the service is up and running healthily.

Half-open state:

Once the state becomes OPEN, we wait for some time in the OPEN state. After a certain period, the state becomes HALF_OPEN.

During this period, we do send some requests to Service to check if we still get the proper response. If the failure rate is below the threshold, the state would become CLOSED. If the failure rate is above the threshold, then the state becomes OPEN once again. This cycle continues till the service becomes stable.

Demo:

https://javatechonline.com/how-to-implement-hystrix-circuit-breaker-in-microservices-application/

For additional info please refer the below link:

https://cloud.spring.io/spring-cloud-netflix/multi/multi__circuit_breaker_hystrix_clients.html

Spring Boot Actuator: Production-ready Features

Spring Boot Actuators module helps us in monitoring and managing our application by providing ready-made features like application's health, audit, HTTP tracing etc.

The recommended way to enable the features is to add a dependency on the spring-boot-starter-actuator 'Starter'.

1. Enabling production ready features:

To add the actuator to a Maven based project, add the following 'Starter' dependency:

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

2. Endpoints:

- Actuator endpoints let you monitor and interact with your application.
- Spring Boot includes several built-in endpoints and lets you add your own
  For example, the health endpoint provides basic application health information.
- Each individual endpoint can be enabled or disabled and exposed (made remotely accessible) over HTTP or JMX.

The built-in endpoints will only be auto configured when they are available. Most applications choose exposure via HTTP, where the ID of the endpoint along with a prefix of /actuator is mapped to a URL. For example, by default, the health endpoint is mapped to /actuator/health.

The following technology-agnostic endpoints are available:

ID	Description
auditevents	Exposes audit events information for the current application. Requires an AuditEventRepository bean.
beans	Displays a complete list of all the Spring beans in your application.
caches	Exposes available caches.
conditions	Shows the conditions that were evaluated on configuration and auto-configuration classes and the reasons why they did or did not match.
configprops	Displays a collated list of all @ConfigurationProperties.
env	Exposes properties from Spring's ConfigurableEnvironment.
flyway	Shows any Flyway database migrations that have been applied. Requires one or more Flyway beans.
health	Shows application health information.
httptrace	Displays HTTP trace information (by default, the last 100 HTTP request-response exchanges). Requires an HttpTraceRepository bean.
info	Displays arbitrary application info.
integrationgraph	Shows the Spring Integration graph. Requires a dependency on spring-integration-core.
loggers	Shows and modifies the configuration of loggers in the application.
liquibase	Shows any Liquibase database migrations that have been applied. Requires one or more Liquibase beans.
metrics	Shows 'metrics' information for the current application.
mappings	Displays a collated list of all @RequestMapping paths.
quartz	Shows information about Quartz Scheduler jobs.

scheduledtasks	Displays the scheduled tasks in your application.
sessions	Allows retrieval and deletion of user sessions from a Spring Session-backed session store. Requires a Servlet-based web application using Spring Session.
shutdown	Let's the application be gracefully shutdown. Disabled by default.
startup	Shows the startup steps data collected by the ApplicationStartup. Requires the SpringApplication to be configured with a BufferingApplicationStartup.
threaddump	Performs a thread dump.

Enabling Endpoints:

By default, all endpoints except for shutdown are enabled. To configure the enablement of an endpoint, use its management.endpoint.<id>.enabled property. The following example enables the shutdown endpoint:

```
management.endpoint.shutdown.enabled=true
```

If you prefer endpoint enablement to be opt-in rather than opt-out, set the management.endpoints.enabled-by-default property to false and use individual endpoint enabled properties to opt back in. The following example enables the info endpoint and disables all other endpoints:

```
management.endpoints.enabled-by-default=false
management.endpoint.info.enabled=true
```

Exposing Endpoints:

Since Endpoints may contain sensitive information, careful consideration should be given about when to expose them. The following table shows the default exposure for the built-in endpoints:

For exposing endpoints, please refer the table in the given link
https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html#actuator.endpoints.exposing

For implementing custom endpoints pls refer
https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html#actuator.endpoints.implementing-custom

https://medium.com/@jamiekee94/enhancing-spring-boot-actuator-with-custom-endpoints-d6343fbaa1ca

For more details on actuators security pls refer the below link
https://docs.spring.io/spring-boot/docs/current/reference/html/actuator.html#actuator.endpoints.security
https://www.amitph.com/how-to-secure-spring-boot-actuator-endpoints/
https://www.devglan.com/spring-security/securing-spring-boot-actuator-endpoints-with-spring-security

For demos
https://www.baeldung.com/spring-boot-actuators
https://www.callicoder.com/spring-boot-actuator/