Fail Faster

Adding Circuit Breakers to your APIs

Craig Freeman, Technical Architect
@craigfreeman
Scenario

99.7% monthly uptime for all the microservices in your e-commerce application

0.3% of 500,000 requests = 1,500 failures
0.3% of hours/month = \(~2.2 \text{ hours of downtime/month}\)

$50 \times 1,500 = $75,000
$75,000/2.2 \text{ hours} = $568/\text{minute}

Every minute counts. We need to fail faster.
Agenda

Background

Circuit breakers

Tutorial

What’s Next?
“In short, the microservice architectural style is an approach to developing a single application as a suite of small services...”

- James Lewis and Martin Fowler
  https://www.martinfowler.com/microservices/
Characteristics of a Microservice Architecture

- Componentization via Services
- Organized around business capabilities
- Products, not Projects
- Smart endpoints and dumb pipes
- Decentralized Governance
- Decentralized Data Management
- Infrastructure Automation
- Design for Failure
- Evolutionary Design
Design for Failure

Enhance the user experience and mitigate loss of revenue in the event of a dependency application error or upstream timeout

Graceful Degradation

Monitoring

Resiliency
Quick responses to API failures means fast user feedback.

Fallbacks minimize network traffic
Monitoring
Streaming metrics/logs for early alerting

What do we measure?

**Latency**
Time it takes for 200 requests to respond
Time it takes for 500 requests to respond

**Traffic**
Rate of requests

**Errors**
Rate of failed requests

**Saturation**
System utilization (CPU usage, memory usage)
- Requires connection to containers
Resiliency

- Container load-balancing/scaling for automatic service restoration
- Fault isolation to prevent cascading failures
Enter : Circuit Breakers

“...an automatically operated electrical switch designed to protect an electrical **circuit** from damage caused by excess current from an **overload** or **short circuit**.”

What are circuit breakers?

Software design pattern used to determine the availability of an upstream service (API, database, etc.).

A circuit breaker should "trip" to stop an application’s requests to a dependent service for a period of time when the error or timeout count for the service exceeds a predefined threshold.

How do they work?

Circuit Breaker States

Closed - Resource has not been tried yet or has been tried and is available

Open - Resource was tried and was unavailable, breaker trips

Half-open - Wait-threshold met, resource was tried again
How do they work?

Closed

Open

Half-Open
How do they work?

Failure Threshold
The circuit will break if the percentage of failing requests exceeds this percentage.

Timeout
How long before the circuit trips if a request takes a long time.

Wait Threshold
How long a circuit should stay broken.
Rolling Window

Previous

Now

1s bucket

10s window
How do they work?

Reference: Hystrix - How It Works
How do I use them?

```javascript
const router = require('express').Router();
const request = require('superagent');
const { sendMetrics } = require('../../utils');

const getUsers = (req, res, next) => request.get('/api/users')
.then(data => res.json(data))
.catch(next);

router.get('/users', getUsers);
module.exports = router;
```
How do I use them?

diff --git a/getUsers.js b/getUsers.js
index 22158d3..3da38c7 100644
--- a/getUsers.js
+++ b/getUsers.js
@@ -1,8 +1,11 @@
 const router = require('express').Router();
 const request = require('superagent');
 const { sendMetrics } = require('./../utils');
+const circuitBreaker = require('opossum');

-const getUsers = (req, res, next) => request.get('/api/users')
+const breaker = circuitBreaker(request.get);
 +
+const getUsers = (req, res, next) => breaker.fire('/api/users')
    .then(data => res.json(data))
    .catch(next);
const router = require('express').Router();
const request = require('superagent');
const { sendMetrics } = require('./utils');
const circuitBreaker = require('opossum');

const breaker = circuitBreaker(request.get, {
  timeout: 3000, // If our function takes longer than 3 seconds, trigger a failure
  errorThresholdPercentage: 50, // When 50% of requests fail, trip the circuit
  resetTimeout: 5000, // After 5 seconds, try again
});

const getUsers = (req, res, next) => breaker.fire('/api/users')
  .then(data => res.json(data))
  .catch(next);

router.get('/users', getUsers);
module.exports = router;
const router = require('express').Router();
const request = require('superagent');
const { sendMetrics } = require('./../utils');
const circuitBreaker = require('opossum');

const breaker = circuitBreaker(request.get, {
  timeout: 3000, // If our function takes longer than 3 seconds, trigger a failure
  errorThresholdPercentage: 50, // When 50% of requests fail, trip the circuit
  resetTimeout: 5000, // After 5 seconds, try again
});

breaker.fallback(() => ({ users: ['Monroe', 'Grant', 'Garfield'] }));

const getUsers = (req, res, next) => breaker.fire('/api/users')
  .then(data => res.json(data))
  .catch(next);

router.get('/users', getUsers);
module.exports = router;
How do I use them?

```javascript
const router = require('express').Router();
const request = require('superagent');
const { sendMetrics } = require('./..//utils');
const circuitBreaker = require('opossum');

const breaker = circuitBreaker(request.get, {
  timeout: 3000, // If our function takes longer than 3 seconds, trigger a failure
  errorThresholdPercentage: 50, // When 50% of requests fail, trip the circuit
  resetTimeout: 5000, // After 5 seconds, try again
});

breaker.fallback(() => ({ users: ['Monroe', 'Grant', 'Garfield'] }));

breaker.on('timeout', () => sendMetrics('users_endpoint.cb.timeout'));
breaker.on('reject', () => sendMetrics('users_endpoint.cb.reject'));
breaker.on('open', () => sendMetrics('users_endpoint.cb.open'));
breaker.on('halfOpen', () => sendMetrics('users_endpoint.cb.half_open'));
breaker.on('close', () => sendMetrics('users_endpoint.cb.close'));
breaker.on('fallback', () => sendMetrics('users_endpoint.cb.fallback'));

const getUsers = (req, res, next) => breaker.fire('/api/users')
  .then(data => res.json(data))
  .catch(next);
Libraries

Hystrix (Java)
Brakes
Opossum
HystrixJS
etc.
Success
Success

Without Circuit Breakers

With Circuit Breakers

Latency for endpoints (in milliseconds) vs. Time (in 10m intervals)

Latency for endpoints w/ circuitbreakers (in milliseconds) vs. Time (in 10m intervals)
Errors

Without Circuit Breakers

With Circuit Breakers
Errors

Without Circuit Breakers

Latency for endpoints (in milliseconds) 10m

With Circuit Breakers

Latency for endpoints w/ circuitbreakers (in milli...) 10m

30 s @ 11:16:00
Timeouts

Without Circuit Breakers

With Circuit Breakers

Diagram showing the difference in handling timeouts with and without circuit breakers.
Timeouts

Without Circuit Breakers

With Circuit Breakers

Latency for endpoints (in milliseconds) 10m

Latency for endpoints w/ circuitbreakers (in milli... 10m
Timeouts

Errors - Rate of failed requests

[Graph showing rate of failed requests with a spike between 16:15 and 16:20.]
Timeouts w/ Fallback

Without Circuit Breakers

With Circuit Breakers
Timeouts w/ Fallback

<table>
<thead>
<tr>
<th>Without Circuit Breakers</th>
<th>With Circuit Breakers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency for endpoints (in milliseconds)</td>
<td>Latency for endpoints w/ circuit breakers (in milliseconds)</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>150</td>
<td>150</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Timeouts w/ Fallback

Without Circuit Breakers

Errors - Rate of failed requests

With Circuit Breakers

Success - Rate of successful requests
Gotchas

Inherently slower

May prevent auto-scaling
Going Forward

Refactor code

Higher Order Function > Decorator
Apply retry pattern or exponential backoff algorithm
Respond with cache in fallback method
Service mesh

Runbooks

Step-by-step instructions for engineers to troubleshoot and solve common issues
Info on metrics/logging dashboards
“On-Call” schedule, etc.

Fine-tuning

Performance testing - Make sure settings don’t prevent automatic load-balancing
Chaos testing (Fire drills) - Refine response to incidents
Fail Faster

Adding Circuit Breakers to your APIs

Resources
- Martin Fowler - Microservices
- Wikipedia - Circuit Breaker Design Pattern
- The 4 Golden Signals of API Health and Performance in Cloud-Native Applications
- NPM - Popular circuit breaker libraries
- Netflix/Hystrix
  - How it Works