µ-Service Resiliency With Circuit Breakers

Lance Ball - Red Hat - @lanceball
FullStack 2018
Resiliency is defined as the capability of a system to maintain its functions and structure in the face of internal and external change and to degrade gracefully when it must.
Microservices
My App

Monolith
Microservices are not a Panacea
A Single Failure
function wait (timeout) {
    return new Promise(resolve => {
        setTimeout(resolve, timeout);
    });
}
const MAX_ATTEMPTS = 10;
let retryAttempts = 0;

async function fetchData (url) {
    return request.get(url).then(formatData)
        .catch(err => {
            if (retryAttempts > MAX_ATTEMPTS) return Promise.reject(err);
            retryAttempts++;
            await wait(500);
            return fetchData(url);
        });
}

A Naive Implementation
What Happens When We Keep On Trying?
Services D & E Block

Diagram showing relationships between different services labeled A to J.
It Gets Worse
Failure Cascades
Unto Death
Naive Implementations are a Band-Aid
Do not have the same error reoccur constantly.

Handle the error quickly and gracefully without waiting for TCP timeout.
Resiliency Patterns

- Fault and latency tolerant
- Stops cascading failures
- Provides fallback behavior
- Fails fast with graduated automatic recovery
Circuit Breakers
function fetchData (url) {
    return _ => {
        // return a promise
        return request.get(url)
            .then(formatData)
            .catch(err => {
                // do something more sensible than this
                console.log(err)
            });
    };
}

A Function: It Might Fail
const CircuitBreaker = require('opossum');

const options = {
    timeout: 1000,
    errorThresholdPercentage: 50,
    resetTimeout: 5000,
    capacity: 10
};

const circuit = CircuitBreaker(fetchData('/some/url'), options);

Wrap it in a Circuit Breaker
Fallback Behavior
circuit.fallback(
    _ => 'Sorry, out of service right now'
);

circuit.on('fallback',
    result => reportFallbackEvent(result));
Events
**fire**  
**success**  
**failure**  
**open**  
**close**  
**halfOpen**  
**fallback**  

**cacheHit**  
**cacheMiss**  
**timeout**  
**semaphore-locked**  
**health-check-failed**  
**snapshot**
Health Checks
```javascript
function memoryUsage () {
    const memory = process.memoryUsage();
    return (memory.heapUsed / memory.heapTotal) < 0.9 ?
        Promise.resolve() : Promise.reject();
}

circuit.healthCheck(memoryUsage);
circuit.on('health-check-failed', sendAlertMessage);

function sendAlertMessage () {
    // send an alert message to someone
}

Example: Health Checks
```
Snapshots
// Creates a 10 second window consisting of ten time slices, 
// each time slice being 1 second long.
const circuit = circuitBreaker(fs.readFile,
{ rollingCountBuckets: 10, rollingCountTimeout: 10000});

// get the cumulative status for the last second
circuit.status.on('snapshot', data => ( /* store data? */ ));

// get the array of 10, 1 second time slices
circuit.status.window;
Statistics
Statistics Snapshots

- Circle color and size represent health and traffic volume.
- Error percentage of last 10 seconds.
- Request rate.
- Circuit-breaker status.
- Last minute latency percentiles.

- Host: 54.0/s
- Cluster: 20,056.0/s
- Circuit Closed

- Hosts
  - Median:
    - 1ms
    - 4ms
  - Mean:
    - 90th
    - 99th
    - 99.5th

- Rolling 10 second counters with 1 second granularity:
  - Successes: 200,545
  - Thread timeouts: 19
  - Thread-pool Rejections: 94
  - Failures/Exceptions: 0
const app = express();
const circuit = CircuitBreaker( callTheRemoteApi );

app.use('/stats.stream', function statStream (request, response) {
  response.writeHead(200, {
    'Content-Type': 'text/event-stream',
    'Cache-Control': 'no-cache',
    'Connection': 'keep-alive'
  });
  circuit.stats.pipe(response);
});
// in the browser
const stats = new EventSource('/stats.stream');
stats.onmessage = updateStats;

function updateStats(message) {
    const stats = JSON.parse(message.data);
    $('#stats').html(message.data);
    $('#failures').html(stats.errorCount);
    $('#fires').html(stats.requestCount);
    $('#latency-mean').html(stats.latencyTotal_mean.toFixed(2));
}
Elizabethan Insults

Demo Time!
Insulting!

POST /api/insult

Circuit Breaker opossum.js

GET /api/adjective

Circuit Breaker opossum.js

GET /api/noun

File system
adjectives.txt

File system
nouns.txt
LAUNCH

Continuous application delivery, built and deployed on OpenShift.

LAUNCH YOUR PROJECT

https://launch.openshift.io
Thanks & Questions

https://launch.openshift.io

https://github.com/bucharest-gold/nodejs-circuit-breaker

https://github.com/lance/elizabethan-insults

https://github.com/bucharest-gold/opossum