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 hours of downtime/month**

$50 x 1,500 = $75,000
$75,000/2.2 hours = $568/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

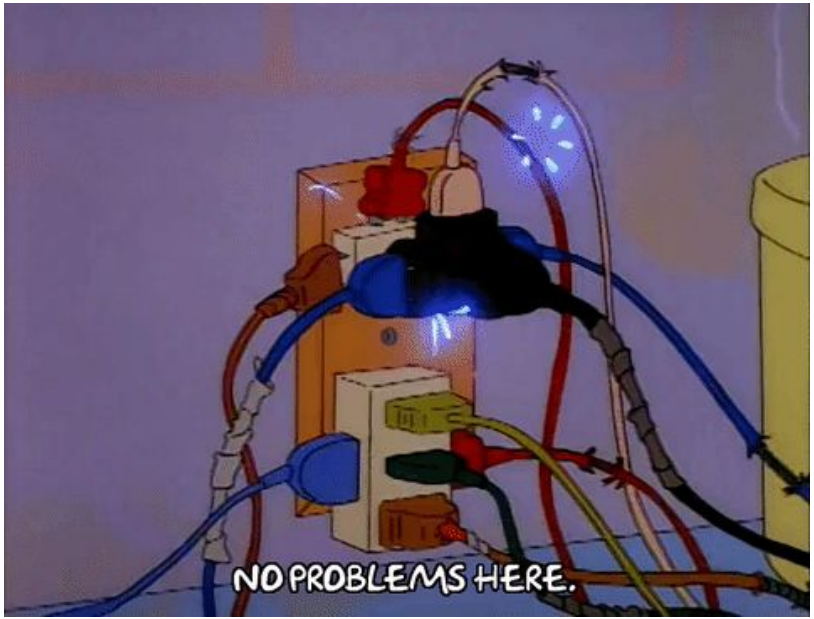
Design for Failure

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

Graceful Degradation

Monitoring

Resiliency



Graceful Degradation

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

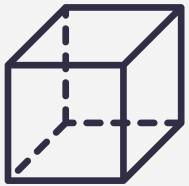
Rate of failed requests

Saturation

System utilization (CPU usage, memory usage)

- Requires connection to containers

Resiliency



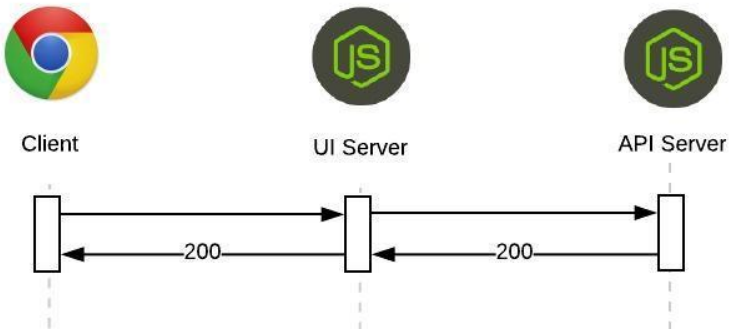
Container load-balancing/scaling for automatic service restoration

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**.”

- Wikipedia, https://en.wikipedia.org/wiki/Circuit_breaker



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~~ Open

How do they work?

Failure Threshold

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

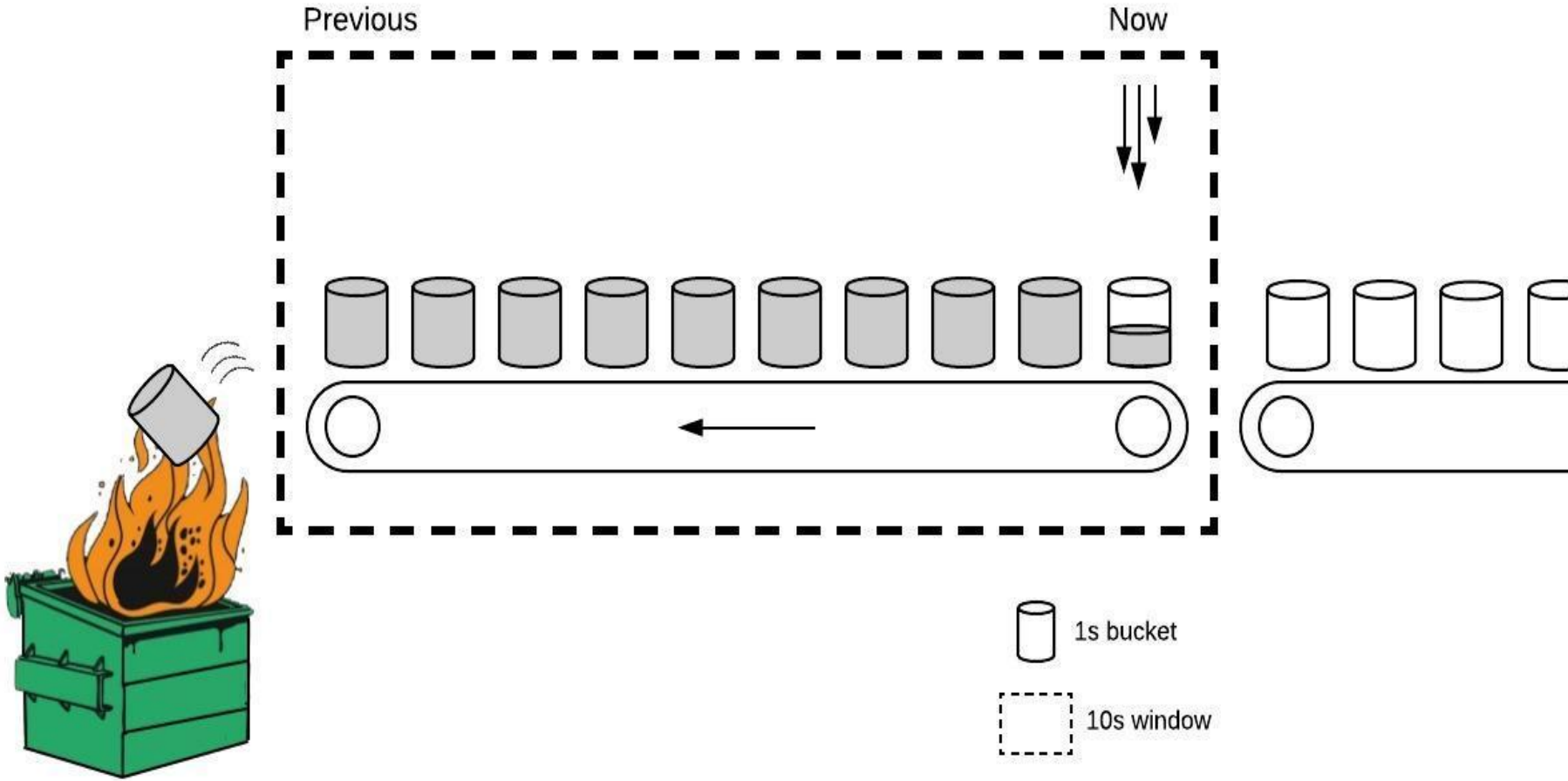
Timeout

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

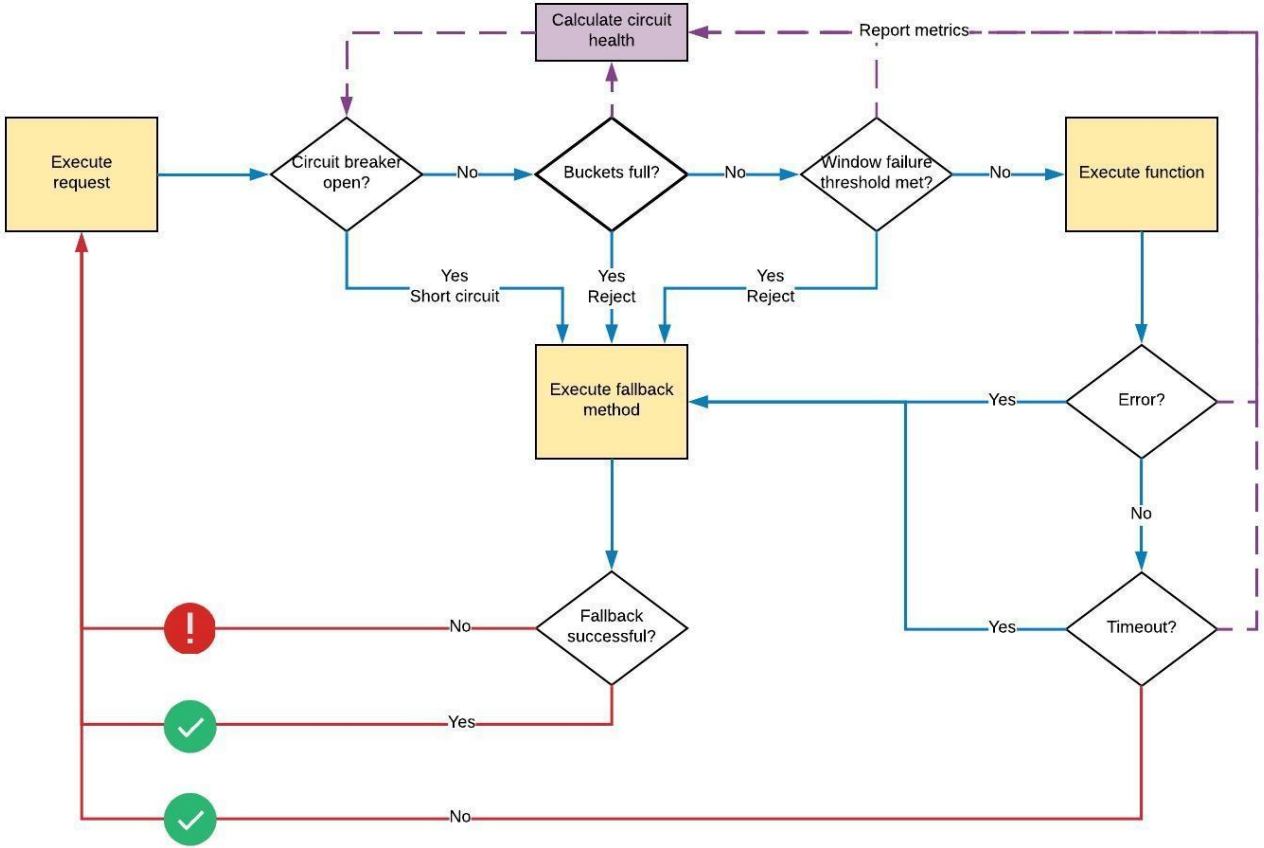
Wait Threshold

How long a circuit should stay broken

Rolling Window



How do they work?



Reference: Hystrix - How It Works

How do I use them?

```
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;

```

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

const breaker = circuitBreaker(request.get);

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?

```
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);
```

How do I use them?

```
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;
```

How do I use them?

```
1  const router = require('express').Router();
2  const request = require('superagent');
3  const { sendMetrics } = require('../utils');
4  const circuitBreaker = require('opossum');
5
6  const breaker = circuitBreaker(request.get, {
7    timeout: 3000,                // If our function takes longer than 3 seconds, trigger a failure
8    errorThresholdPercentage: 50, // When 50% of requests fail, trip the circuit
9    resetTimeout: 5000,           // After 5 seconds, try again
10 });
11
12 breaker.fallback(() => ({ users: ['Monroe', 'Grant', 'Garfield'] }));
13
14 const getUsers = (req, res, next) => breaker.fire('/api/users')
15   .then(data => res.json(data))
16   .catch(next);
17
18 router.get('/users', getUsers);
19 module.exports = router;
20
```

How do I use them?

```
1  const router = require('express').Router();
2  const request = require('superagent');
3  const { sendMetrics } = require('../utils');
4  const circuitBreaker = require('opossum');
5
6  const breaker = circuitBreaker(request.get, {
7    timeout: 3000,                // If our function takes longer than 3 seconds, trigger a failure
8    errorThresholdPercentage: 50, // When 50% of requests fail, trip the circuit
9    resetTimeout: 5000,           // After 5 seconds, try again
10 });
11
12 breaker.fallback(() => ({ users: ['Monroe', 'Grant', 'Garfield'] }));
13
14 breaker.on('timeout', () => sendMetrics('users_endpoint.cb.timeout'));
15 breaker.on('reject', () => sendMetrics('users_endpoint.cb.reject'));
16 breaker.on('open', () => sendMetrics('users_endpoint.cb.open'));
17 breaker.on('halfOpen', () => sendMetrics('users_endpoint.cb.half_open'));
18 breaker.on('close', () => sendMetrics('users_endpoint.cb.close'));
19 breaker.on('fallback', () => sendMetrics('users_endpoint.cb.fallback'));
20
21 const getUsers = (req, res, next) => breaker.fire('/api/users')
22   .then(data => res.json(data))
```

Libraries

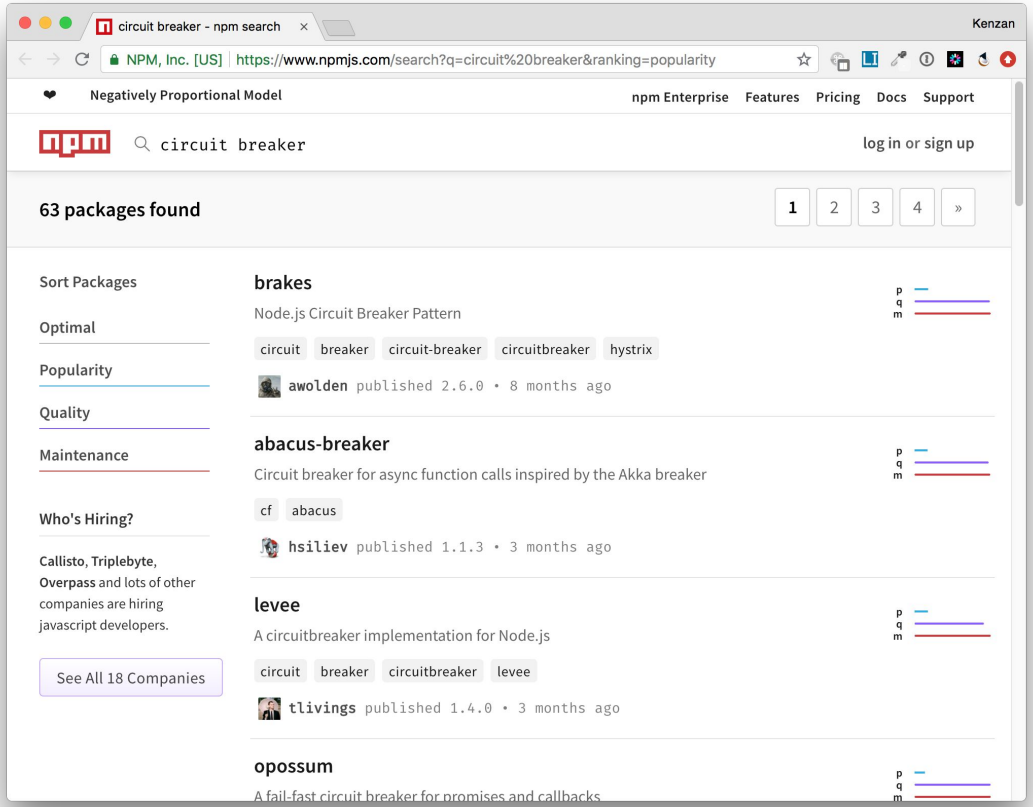
Hystrix (Java)

Brakes

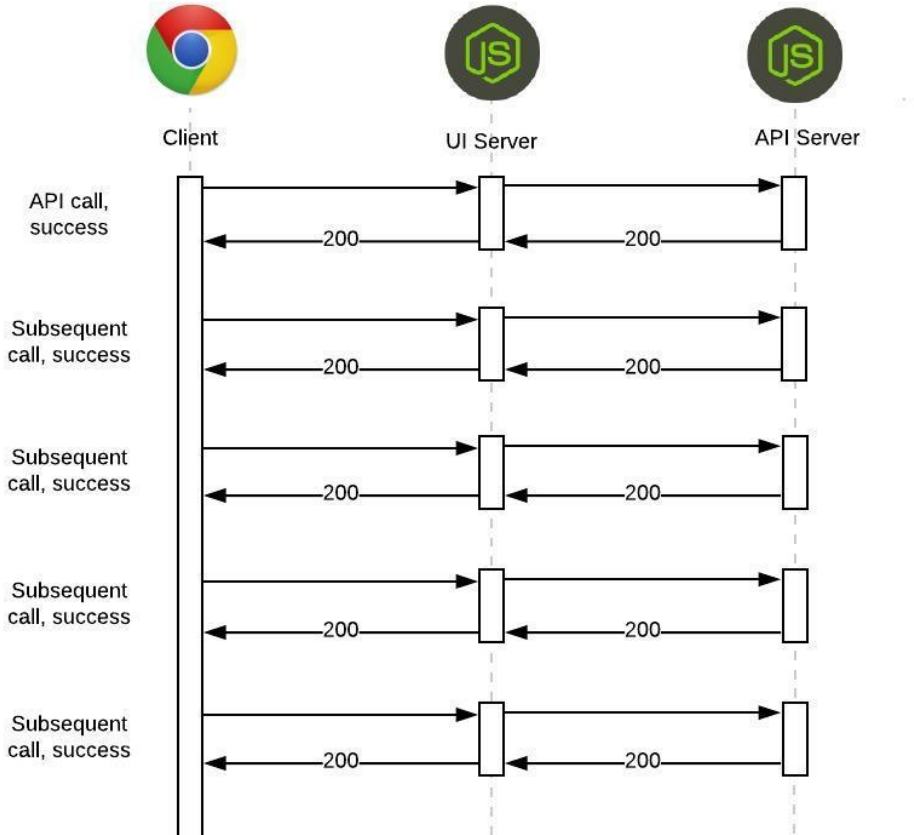
Opossum

HystrixJS

etc.

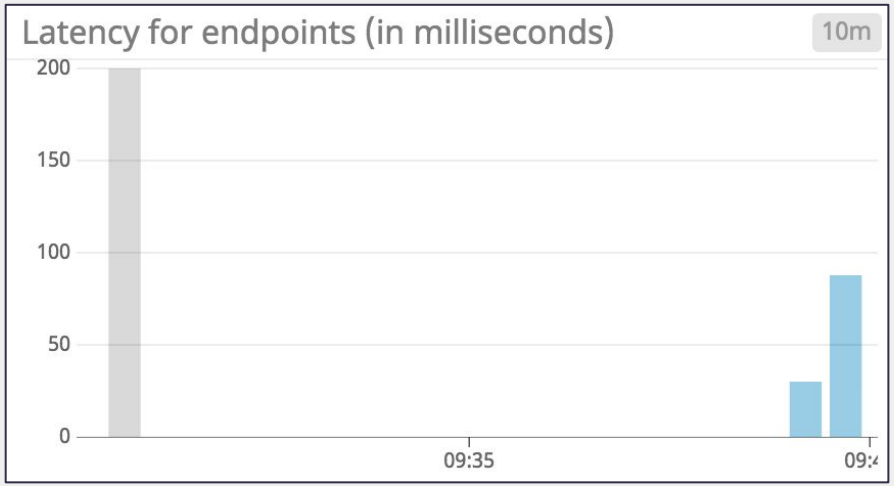


Success

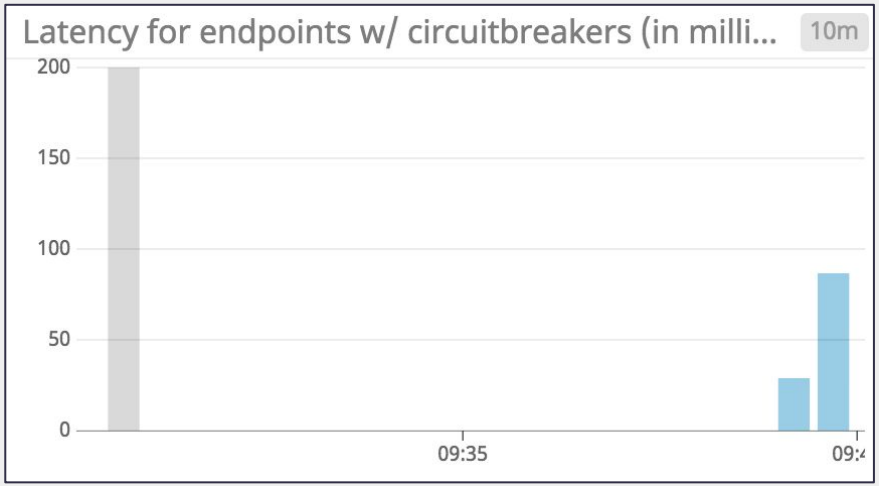

Client
UI Server
API Server
API call, success
200
200
Subsequent call, success
200
200
Subsequent call, success
200
200
Subsequent call, success
200
200
Subsequent call, success
200
200

Success

Without Circuit Breakers

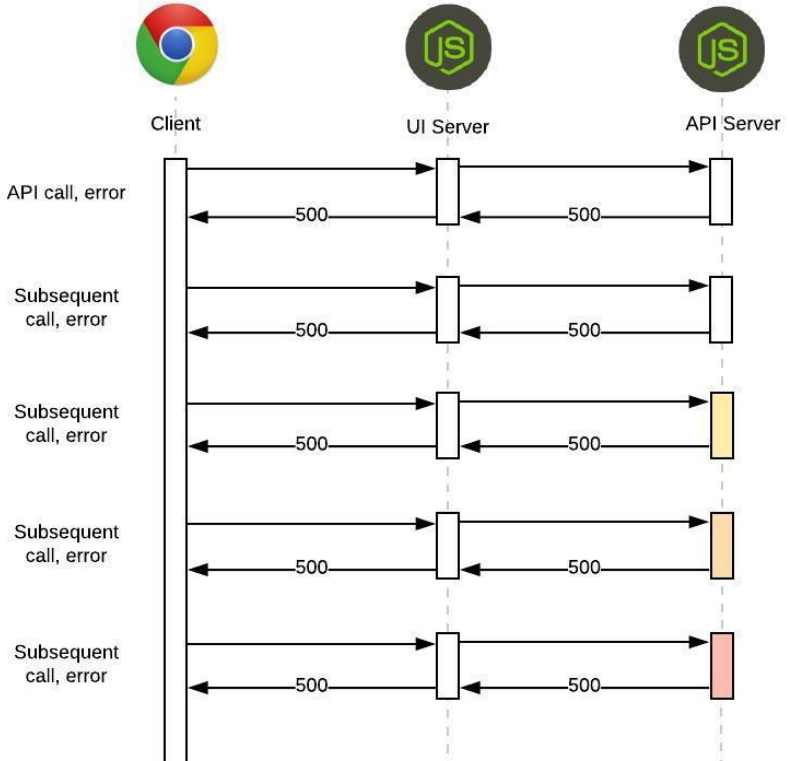


With Circuit Breakers

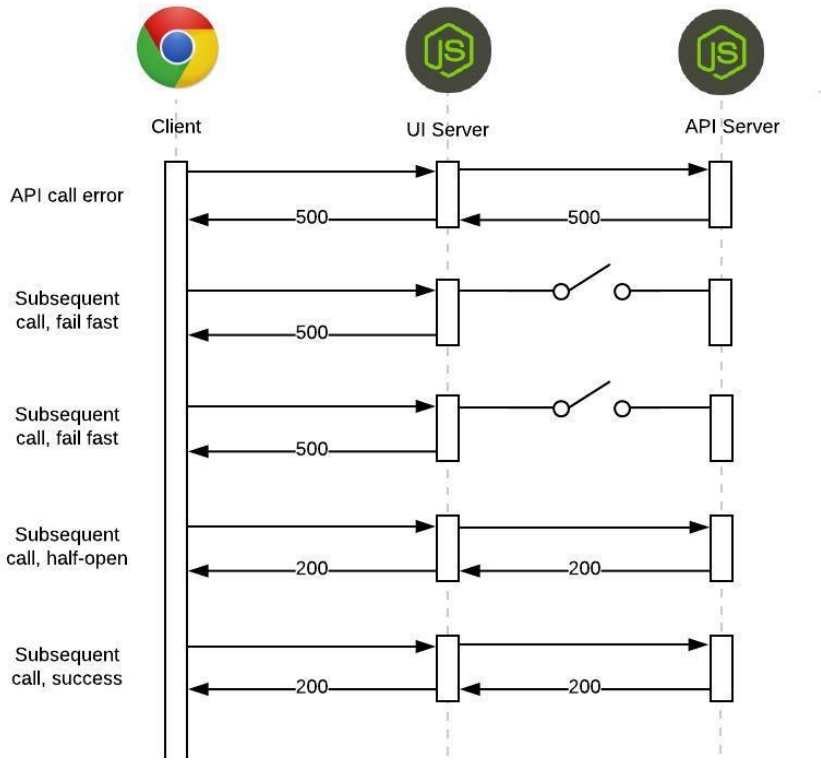


Errors

Without Circuit Breakers

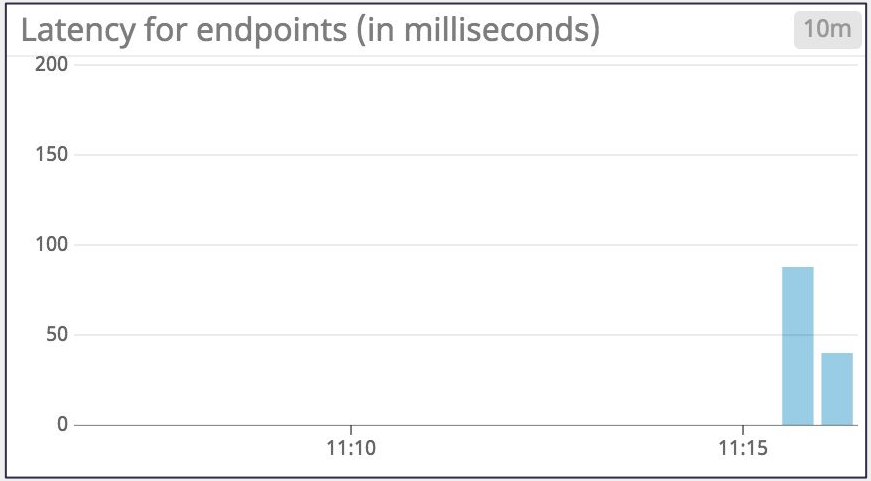


With Circuit Breakers

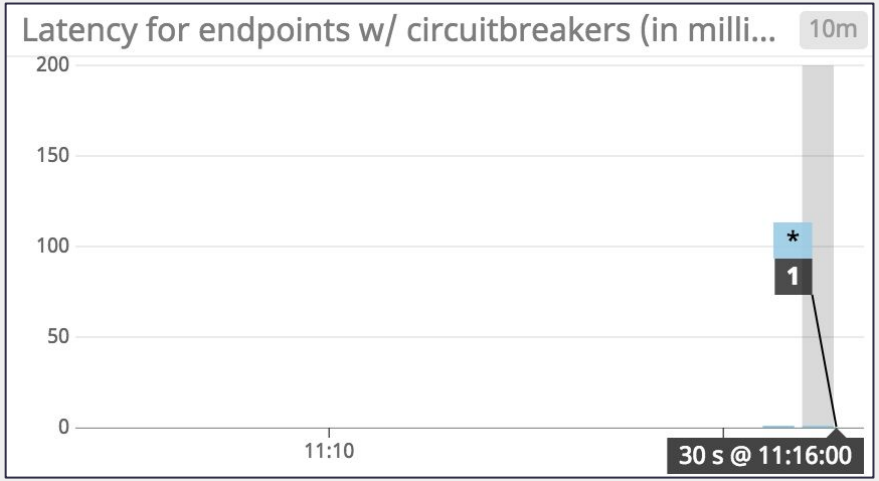


Errors

Without Circuit Breakers

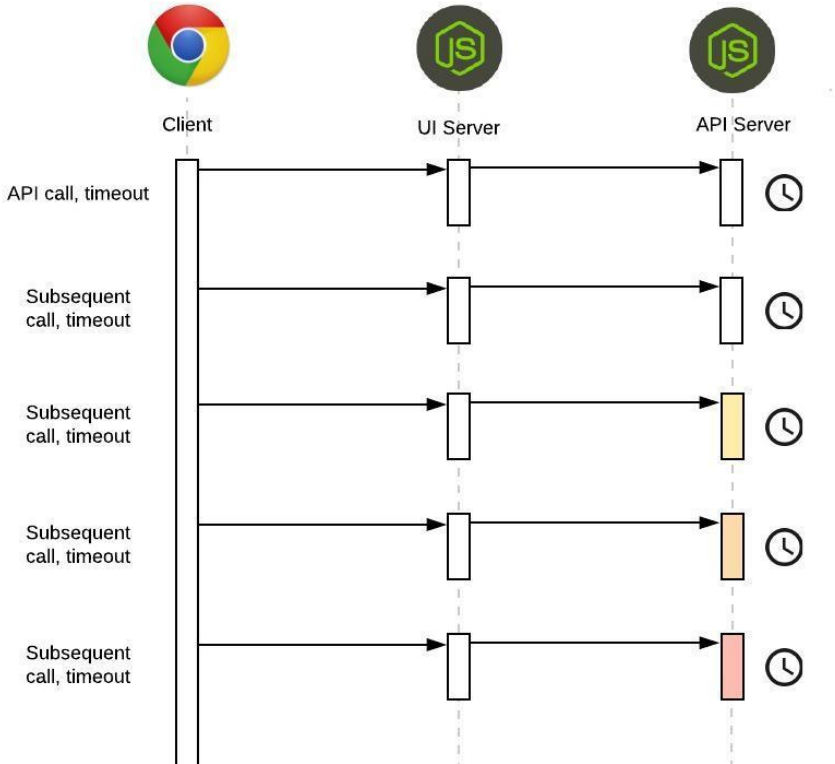


With Circuit Breakers

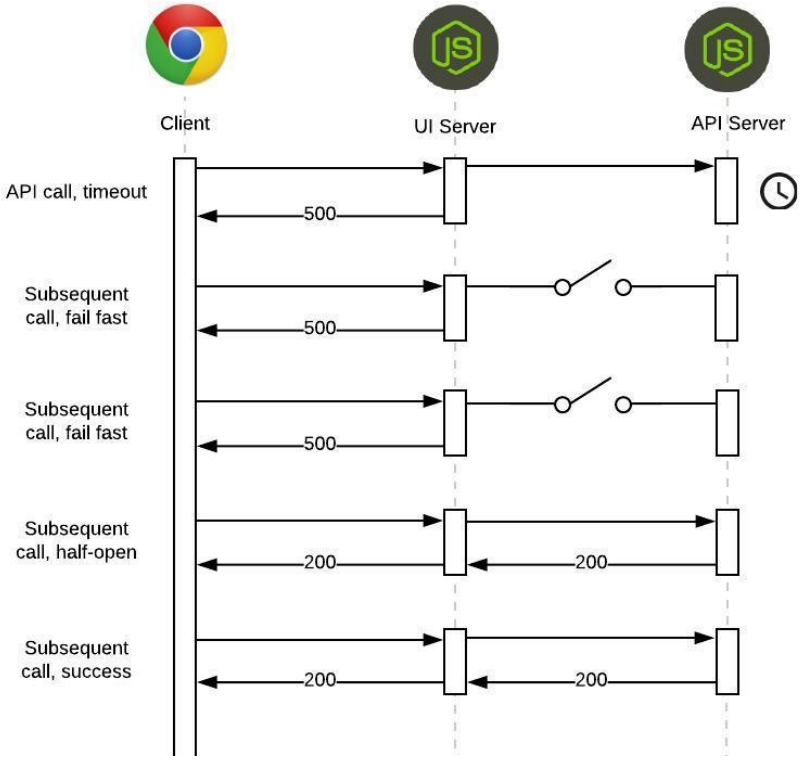


Timeouts

Without Circuit Breakers

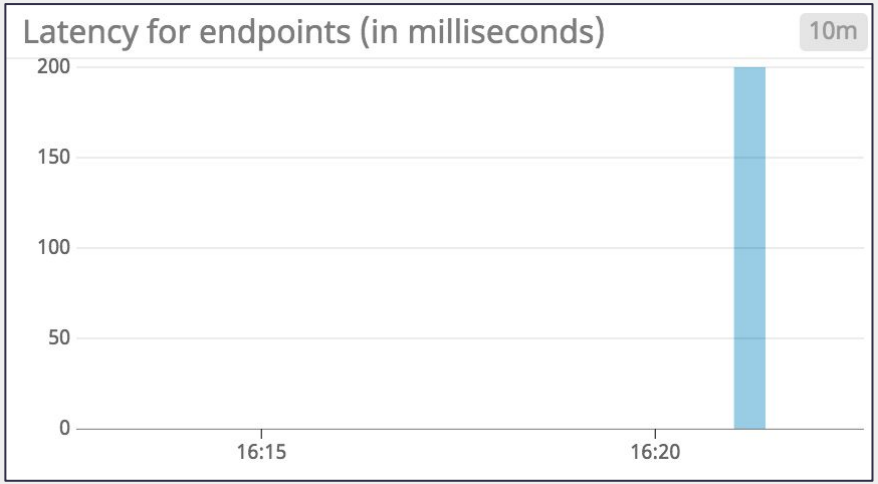


With Circuit Breakers

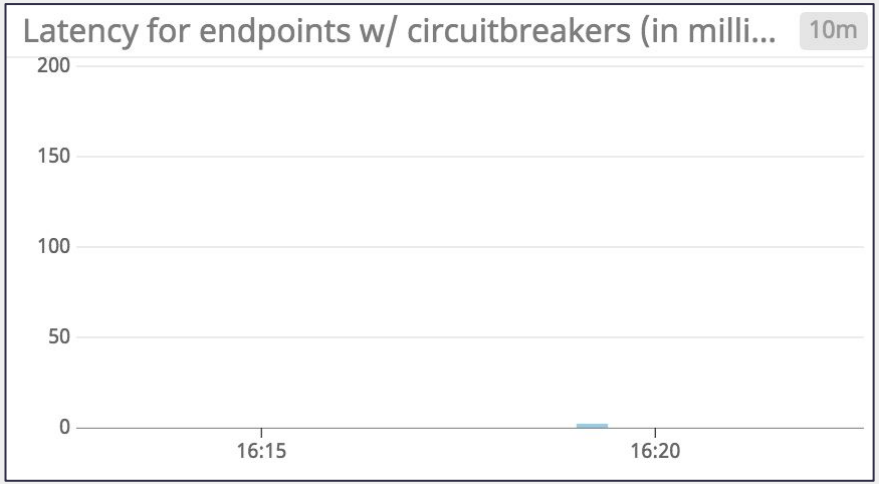


Timeouts

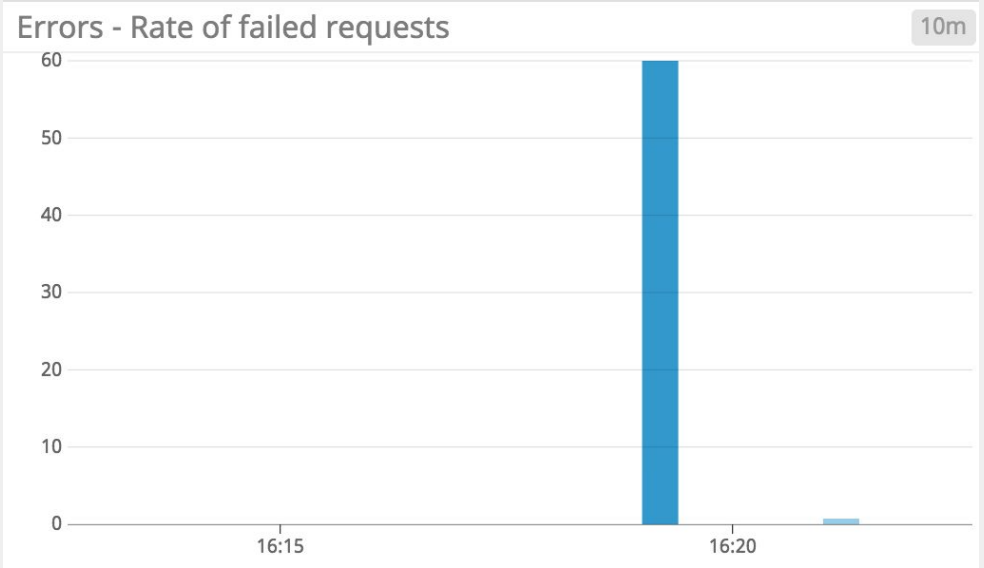
Without Circuit Breakers



With Circuit Breakers

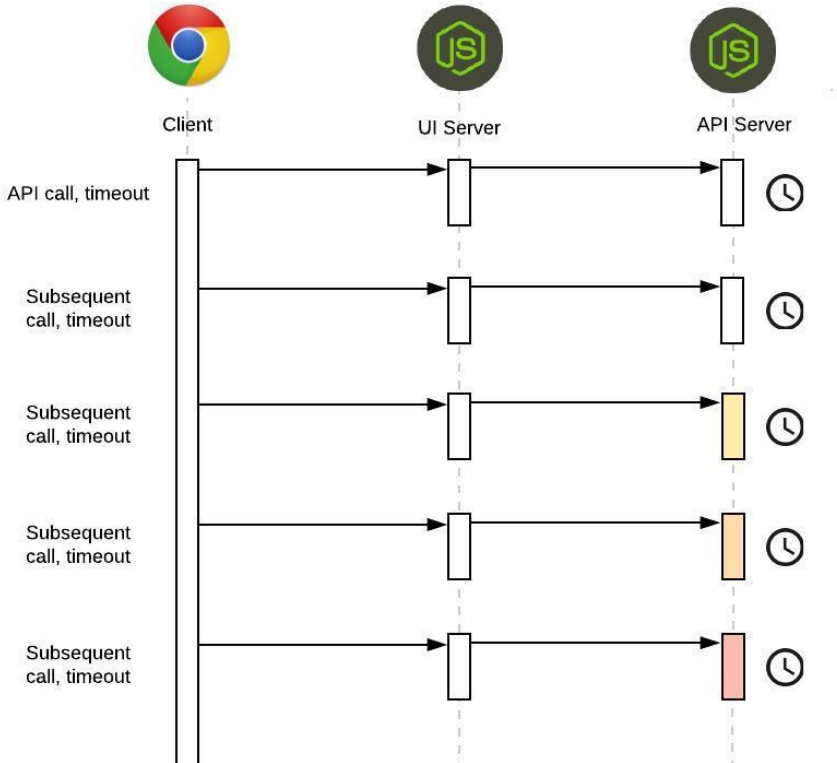


Timeouts

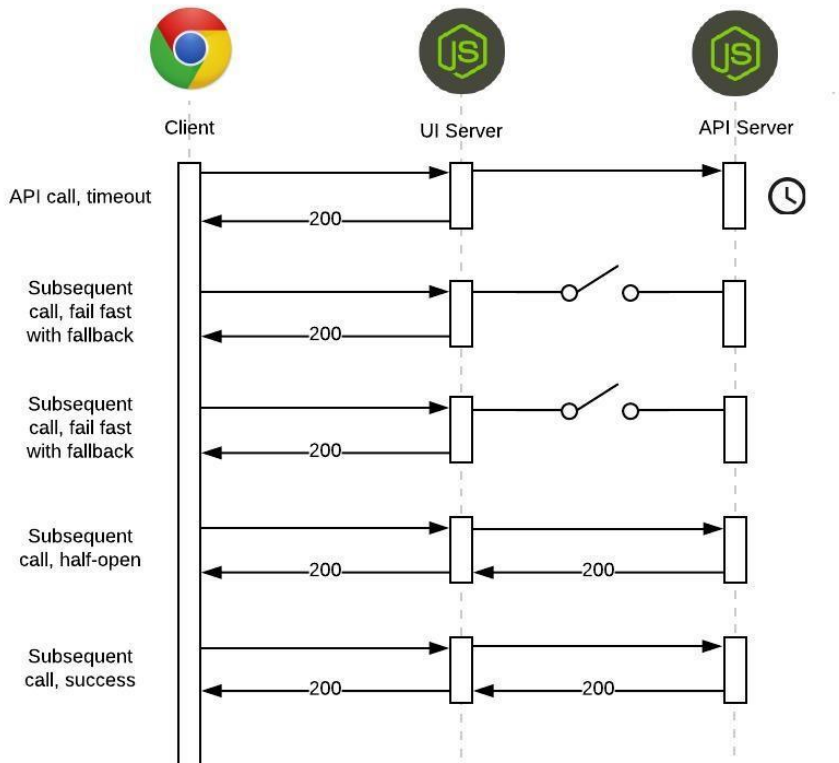

Errors - Rate of failed requests
10m
60
50
40
30
20
10
0
16:15
16:20

Timeouts w/ Fallback

Without Circuit Breakers

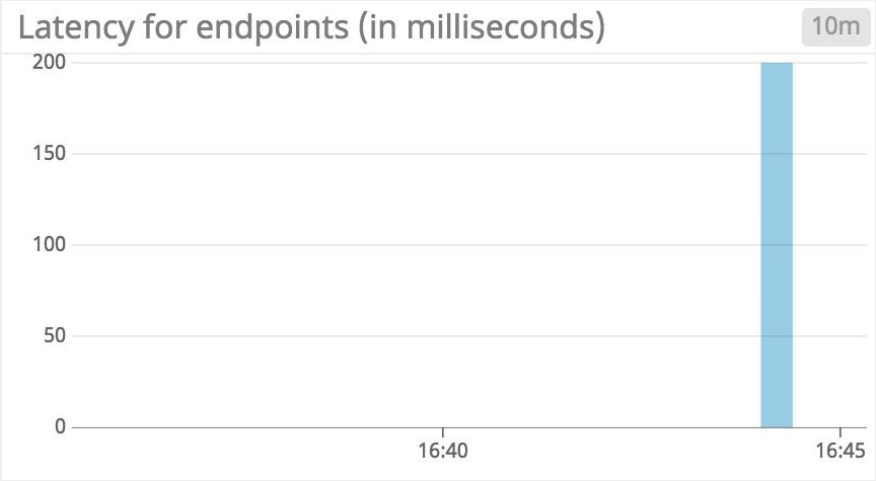


With Circuit Breakers

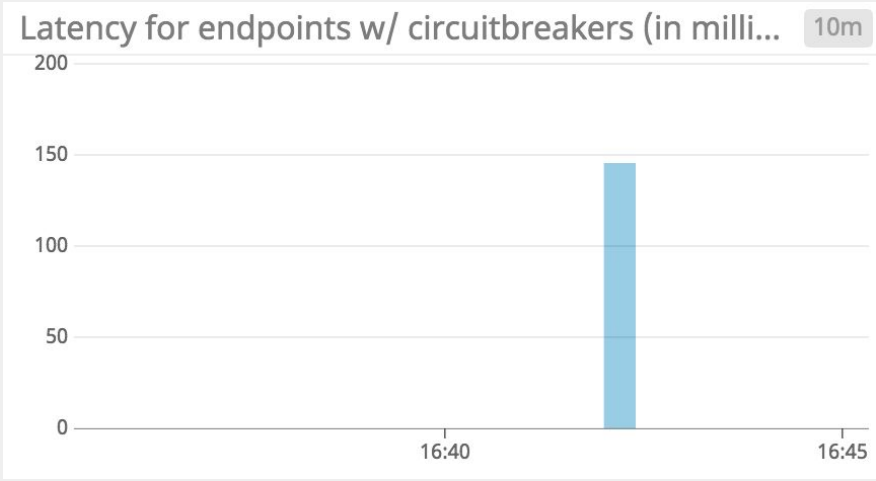


Timeouts w/ Fallback

Without Circuit Breakers



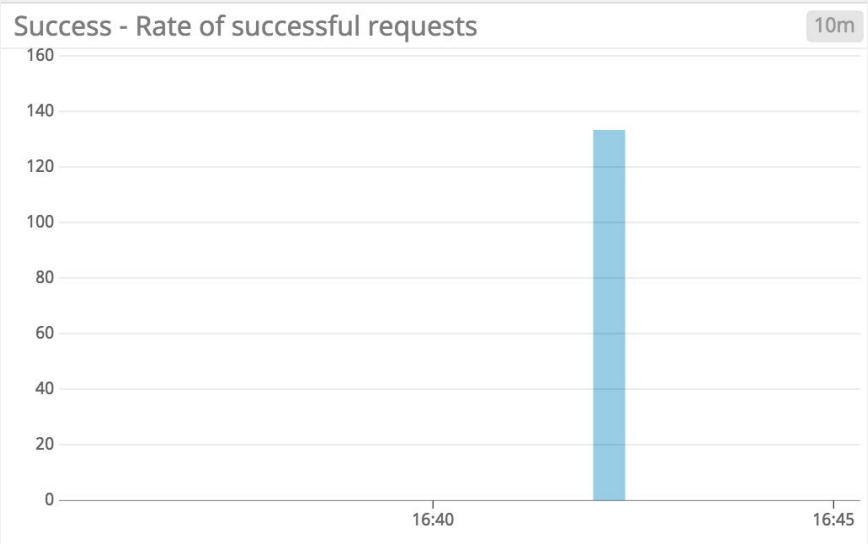
With Circuit Breakers



Timeouts w/ Fallback

Without Circuit Breakers

With Circuit Breakers



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

Craig Freeman, Kenzan
@craigfreeman
craigfreeman.net

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