

# **μ-Service Resiliency With Circuit Breakers**

---

Lance Ball - Red Hat - @lanceball  
FullStack 2018



# Resilience

*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.*

TOWARD INHERENTLY SECURE AND RESILIENT SOCIETIES

Brad Allenby, Jonathan Fink

<http://science.sciencemag.org/content/309/5737/1034.full>

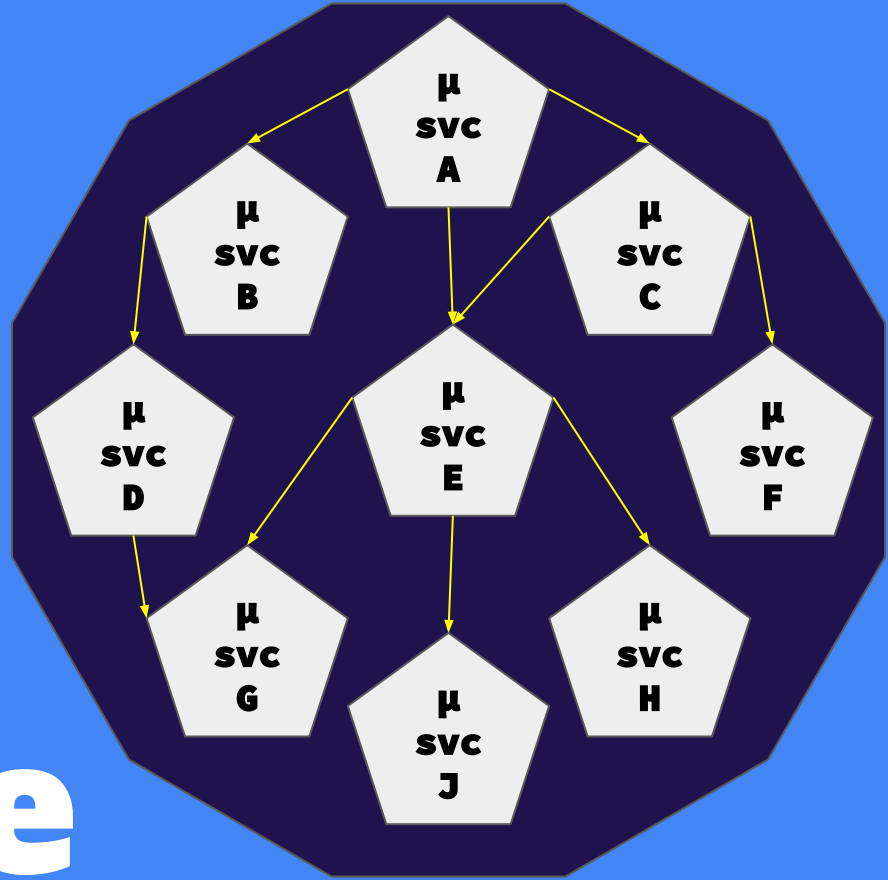
# Microservices

**My App**

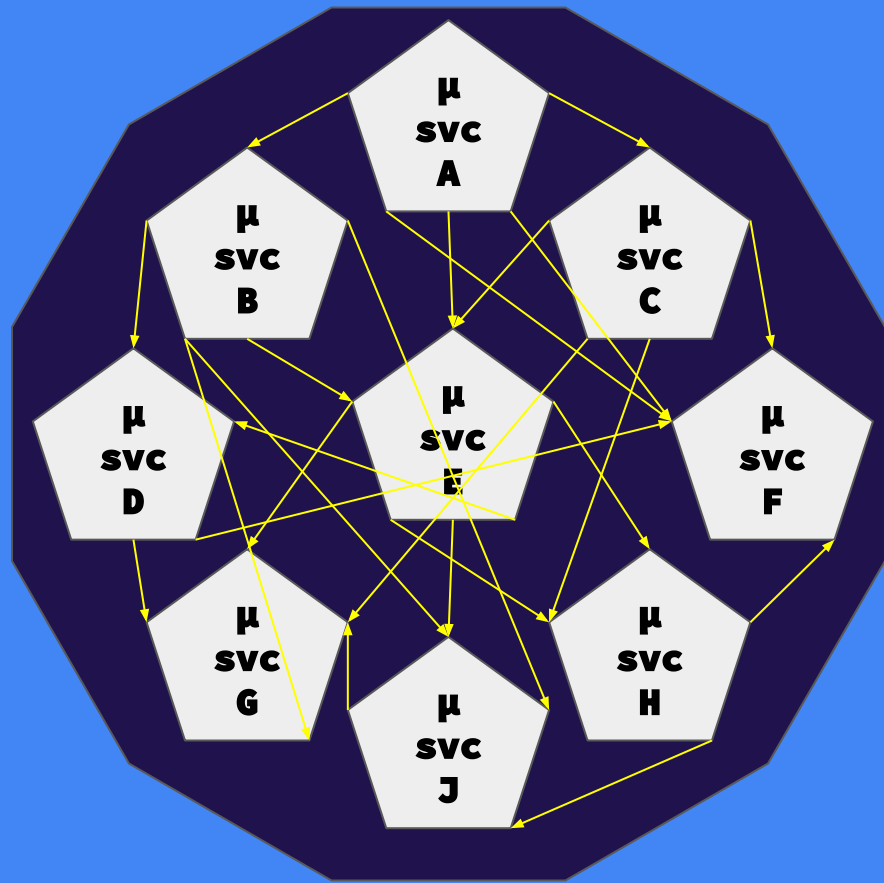


**Monolith**

# $\mu$ -Service

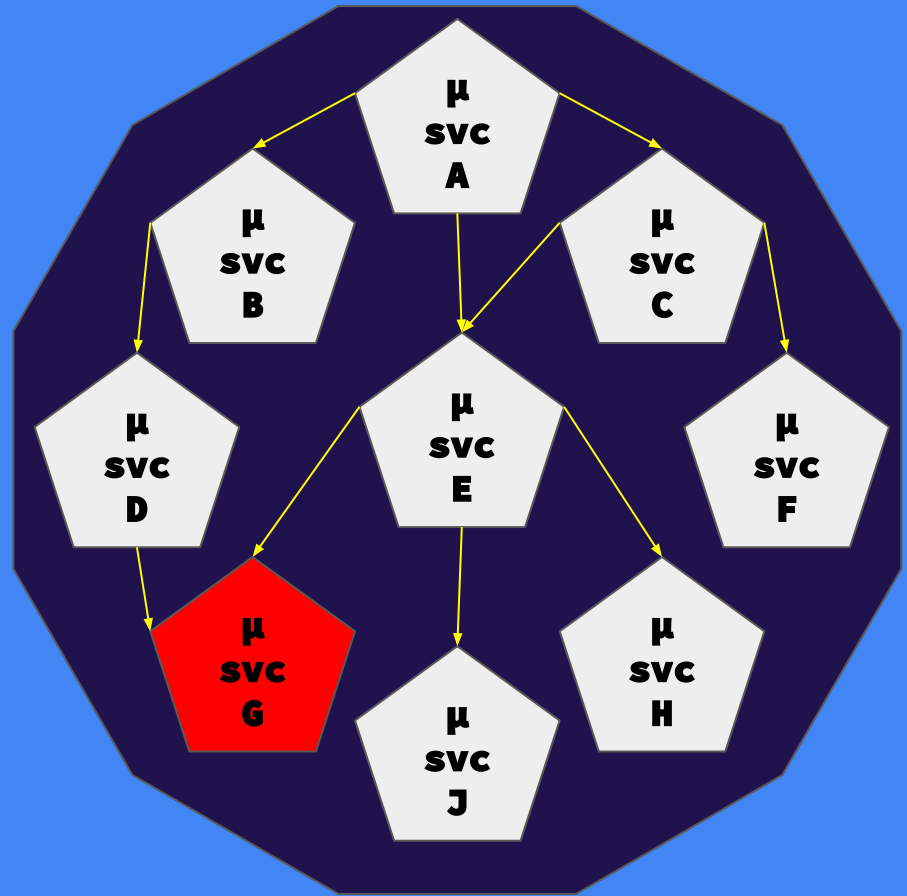


# Reality



# **Microservices are not a Panacea**

# A Single Failure





```
function wait (timeout) {  
  return new Promise(resolve => {  
    setTimeout(resolve, timeout)  
  });  
}
```

**A Simple Sleep in  $\mu$ -Service D**

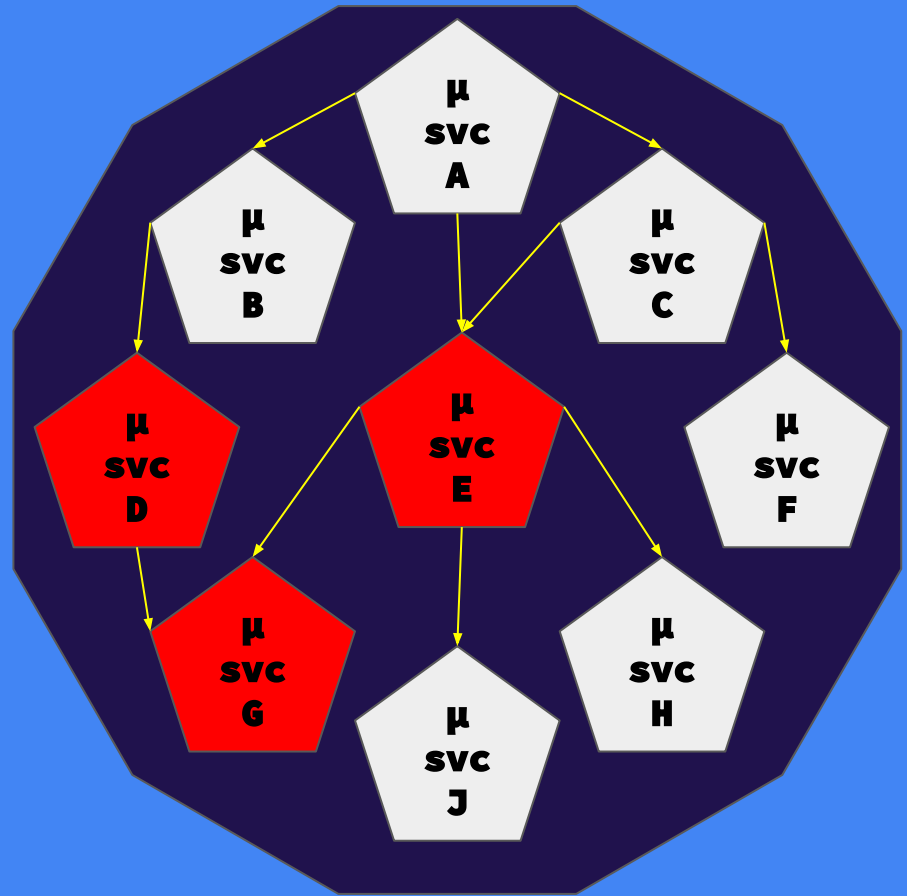
```
const MAX_ATTEMPTS = 10;
let retryAttempts = 0;

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

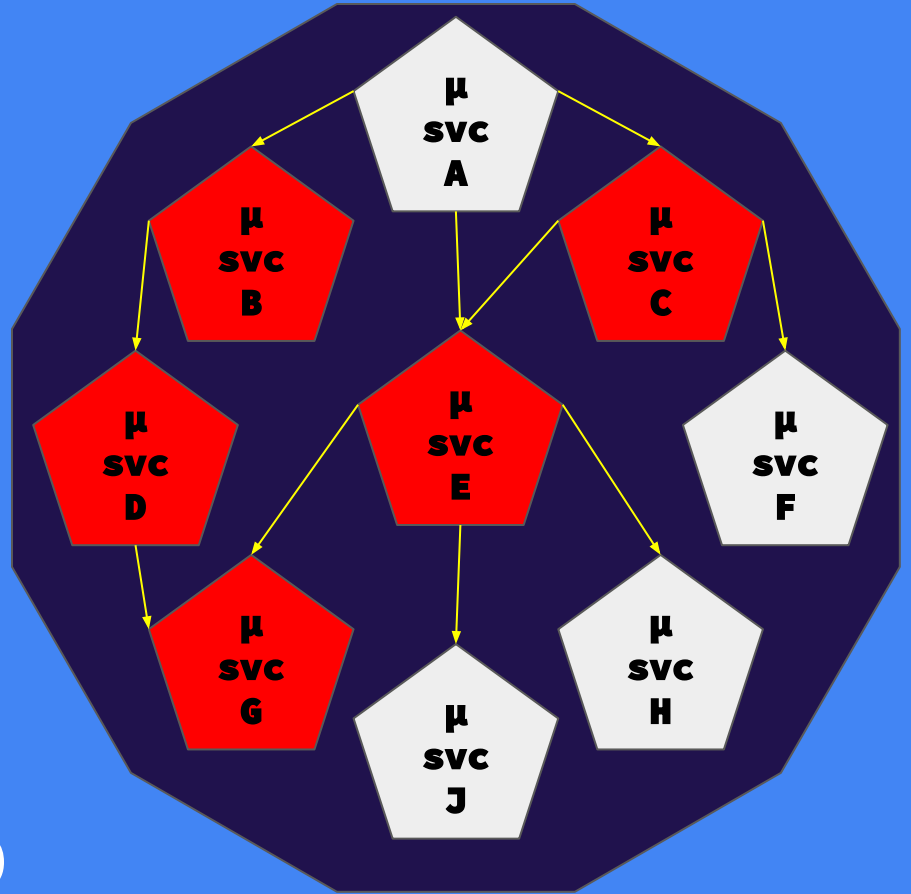
**What Happens When  
We Keep On Trying?**

# Services D & E Block

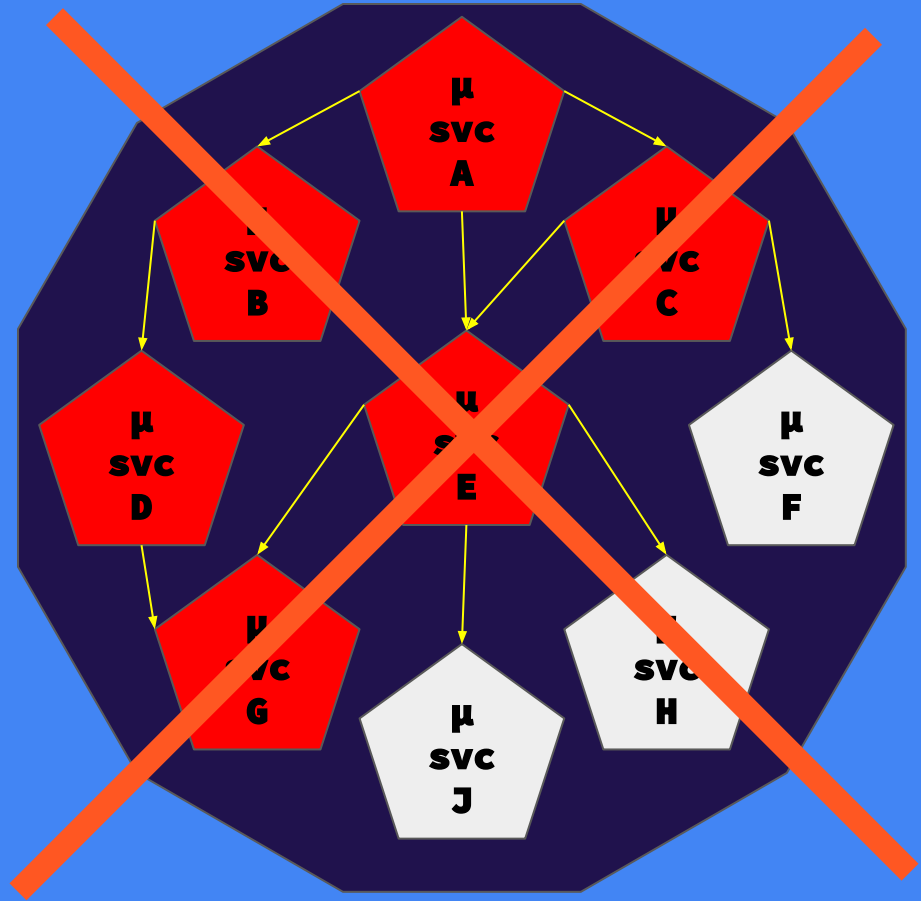


**It Gets Worse**

# Failure Cascades



# Unto Death



**Naive Implementations  
are a Band-Aid**



# Resilience

Do not have the same error reoccur constantly.

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

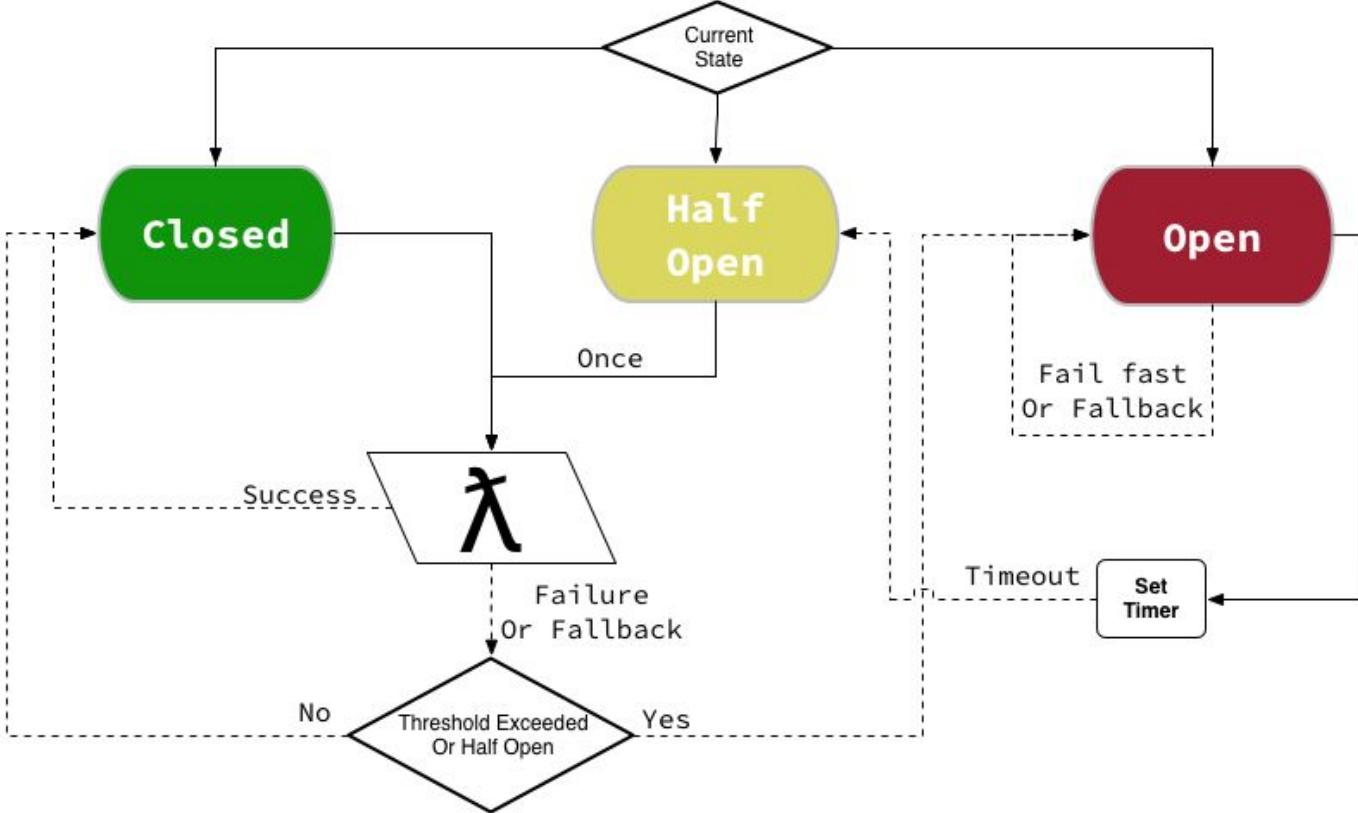
# Resiliency Patterns

- Fault and latency tolerant
- Stops cascading failures
- Provides fallback behavior
- Fails fast with graduated automatic recovery

---

# Circuit Breakers

# Circuit Breaker



```
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));
```

## **Fallback Events**



# Events

- ★ fire
- ★ success
- ★ failure
- ★ open
- ★ close
- ★ halfOpen
- ★ fallback

- ★ cacheHit
- ★ cacheMiss
- ★ timeout
- ★ semaphore-locked
- ★ health-check-failed
- ★ snapshot

## Events

# Health Checks

```
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

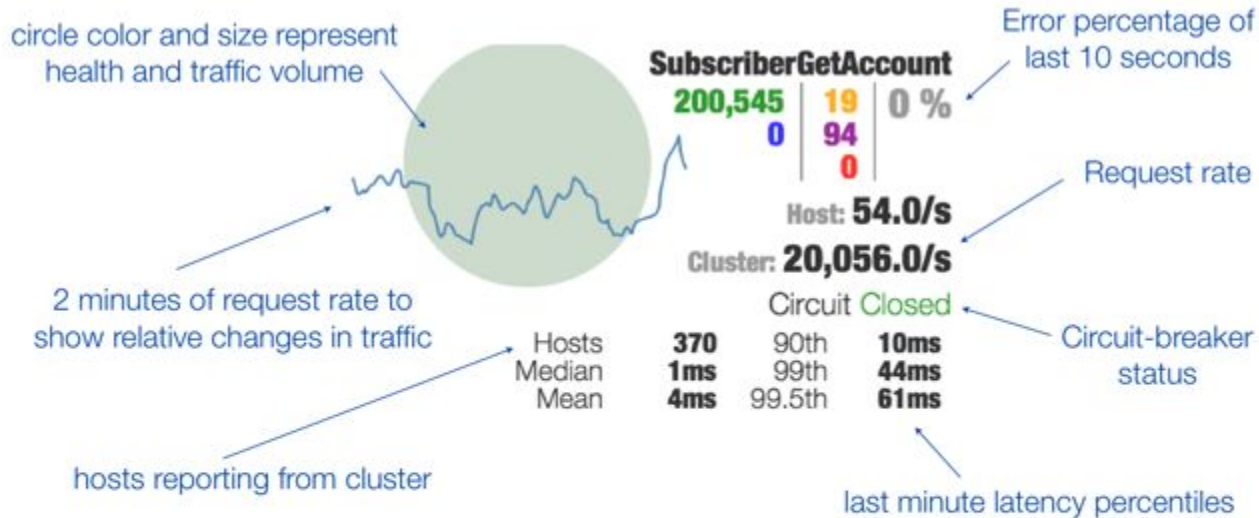
# Snapshots

```
// 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;
```

## Example: Snapshots

**Statistics**

# Statistics Snapshots



Rolling 10 second counters  
with 1 second granularity

Successes	200,545	19	Thread timeouts
Short-circuited (rejected)	0	94	Thread-pool Rejections
		0	Failures/Exceptions



```
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);
});
```

## Statistics Stream on the Server

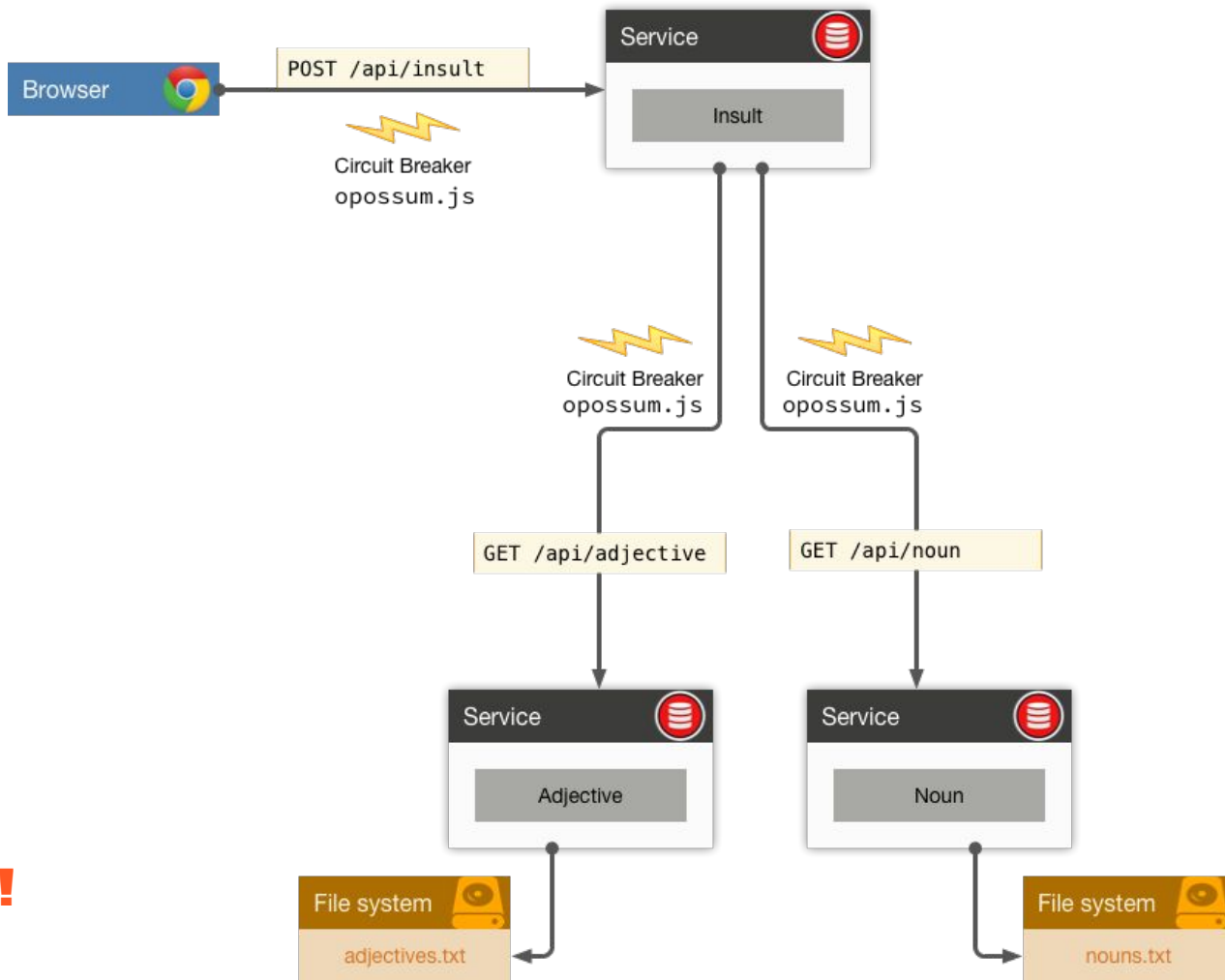
```
// in the browser
const stats = new EventSource('/stats.stream');
stats.onmessage = updateStats;

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));
}
```

## Statistics Stream in the Browser

# Elizabethan Insults

Demo Time!



**Insulting!**

# 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>