Consuming New Node.js Observability Features in a Kubernetes Environment

NodeConf.EU Remote 2021
October 19, 2021

Lucas Holmquist
Sr. Software Engineer
Red Hat

Red Hat

Who Am I - Luke

Sr. Software Engineer @ Red Hat

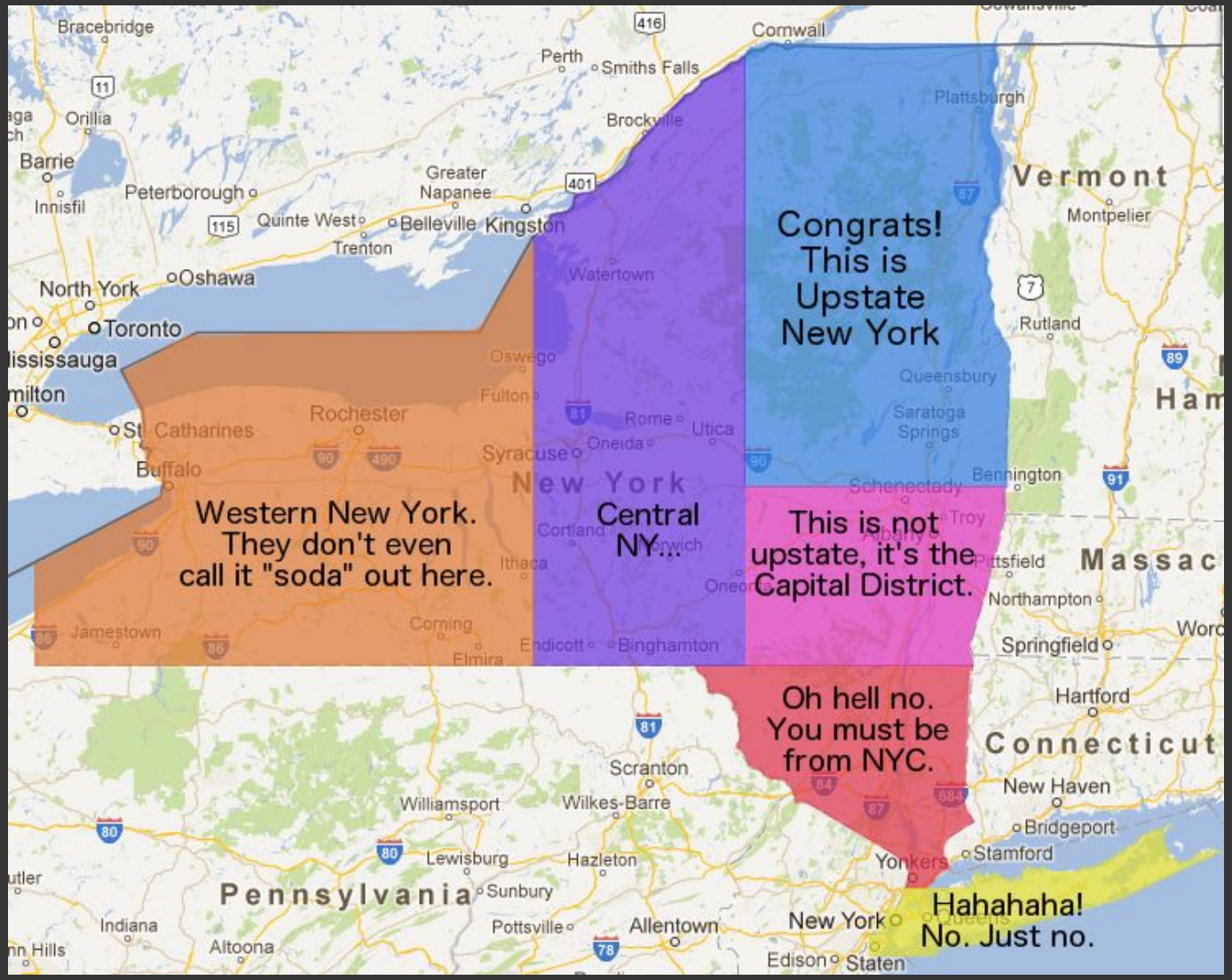
Nodeshift Project

@sienaluke on Twitter

Current Star Wars Trivia Title Holder

Phish Phan

My Location



Red Hat

Congrats!

Red Hat

Red Hat

Agenda

- Node.js Key Observibility Metrics
- Access Data from Kubernetes
- Metrics to Collect
- Demo

What is Observibility?

"A measure of how well our application is running can using some metrics that our runtime provides."uts"

Source: wikipedia - https://en.wikipedia.org/wiki/Observability

Runtime Metrics

Console.log

- Everyone's favorite

```
'use strict';

function funtimes () {
 console.log('HERE');
}

funtimes();
```

Profiling

- Profiler inside V8
- Stack Samples
- --prof
- --prof_process

```
$ node --prof app.js

$ node --prof_process v8.log > output.txt
```

Profiling

```
 ticks parent  name
19557   51.8%  node::crypto::PBKDF2(v8::FunctionCallbackInfo<v8::Value> const&)
19557  100.0%    v8::internal::Builtins::~Builtins()
19557  100.0%      LazyCompile: ~pbkdf2 crypto.js:557:16

 4510   11.9%  _sha1_block_data_order
 4510  100.0%    LazyCompile: *pbkdf2 crypto.js:557:16
 4510  100.0%      LazyCompile: *exports.pbkdf2Sync crypto.js:552:30

 3165    8.4%  _malloc_zone_malloc
 3161   99.9%    LazyCompile: *pbkdf2 crypto.js:557:16
 3161  100.0%      LazyCompile: *exports.pbkdf2Sync crypto.js:552:30
```


https://nodejs.org/en/docs/guides/simple-profiling/

Profiling

- Inspector api
- Upload to Chrome Dev Tools

```
const inspector = require('inspector');
const fs = require('fs');
const session = new inspector.Session();
session.connect();

session.post('Profiler.enable', () => {
 session.post('Profiler.start', () => {
   // Invoke business logic under measurement here...

   // some time later...
   session.post('Profiler.stop', (err, { profile }) => {
     // Write profile to disk, upload, etc.
     if (!err) {
       fs.writeFileSync('./profile.cpuprofile', JSON.stringify(profile));
     }
   });
 });
});
```


https://nodejs.org/api/inspector.html#inspector_example_usage

Trace Events

- V8, Node Core, User Code
- CLI Flags
- Module
- Chrome Dev Tools

```
const trace_events = require('trace_events');
const tracing = trace_events.createTracing({ categories:
['node.promise.rejections] });
tracing.enable();  // Enable trace event capture for the
'node.promise.rejections' category

// do work

tracing.disable();  // Disable trace event capture for the
'node.promise.rejections' category
```


https://nodejs.org/dist/latest-v14.x/docs/api/tracing.html

Perf Hooks

- Performance Measurements
- Subset of Web Perf APIs
- Measure Asynchronous Code
- Event Loop Utilization(ELU)

```
const { PerformanceObserver, performance } =
require('perf_hooks');

const obs = new PerformanceObserver((items) => {
 console.log(items.getEntries()[0].duration);
 performance.clearMarks();
});
obs.observe({ entryTypes: ['measure'] });
performance.measure('Start to Now');

performance.mark('A');
doSomeLongRunningProcess(() => {
 performance.measure('A to Now', 'A');

 performance.mark('B');
 performance.measure('A to B', 'A', 'B');
});
```


https://nodejs.org/dist/latest-v14.x/docs/api/perf_hooks.html

Perf Hooks

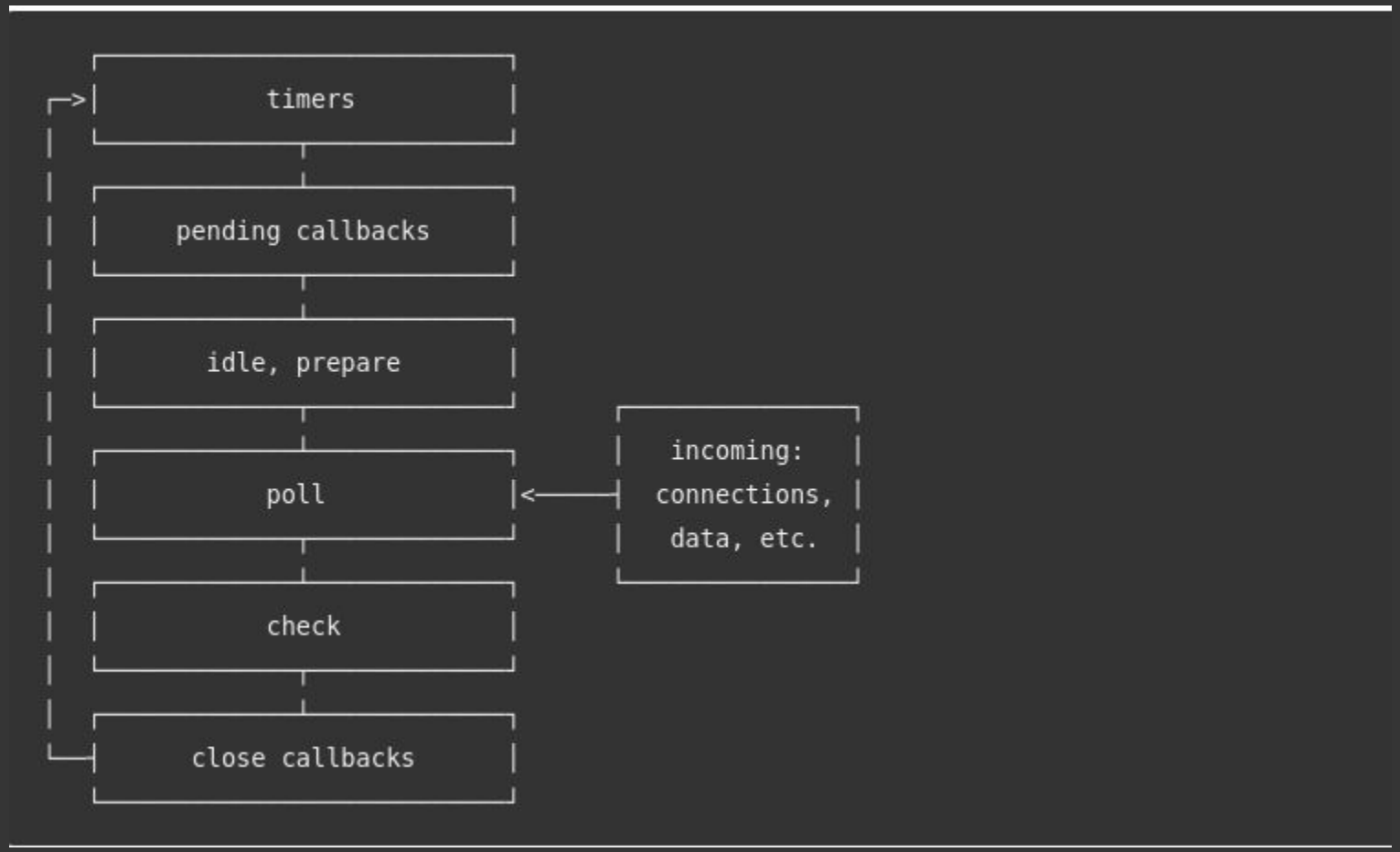
```
'use strict';
const {
 performance,
 PerformanceObserver
} = require('perf_hooks');
const mod = require('module');

// Monkey patch the require function
mod.Module.prototype.require =
 performance.timerify(mod.Module.prototype.require);
require = performance.timerify(require);

// Activate the observer
const obs = new PerformanceObserver((list) => {
 const entries = list.getEntries();
 entries.forEach((entry) => {
   console.log(`require('${entry[0]}')`, entry.duration);
 });
 obs.disconnect();
});
obs.observe({ entryTypes: ['function'], buffered: true });

require('some-module');
```

Event Loop



https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

Perf Hooks - ELU

- Added in
  - V14.10.0, v12.19.0
- Event Loop Stats, Not CPU

```
'use strict';
const { eventLoopUtilization } =
require('perf_hooks').performance;

const { spawnSync } = require('child_process');

setImmediate(() => {
 const elu = eventLoopUtilization();
 spawnSync('sleep', ['5']);
 console.log(eventLoopUtilization(elu));
});
```

Getting Access From A Container Platform

Prometheus

Prometheus is an installable service that gathers instrumentation metrics from your applications and stores them as time-series data.

- Dimensional Data
- Powerful Queries
- Great Visualization
- Efficient Storage
- Many Client Libraries

Prom-client

A prometheus client for Node.js that supports histogram, summaries, gauges and counters.

(https://www.npmjs.com/package/prom-client ~ 770,283 downloads / week)

- Default Metrics
- Custom Metrics
- Multiple Registries
- Pushgateway
- Garbage Collection Metrics

Prom-client

A small code snippet on how we use prom-client

```
const client = require('prom-client');
const collectDefaultMetrics = client.collectDefaultMetrics;
collectDefaultMetrics({ prefix: 'my_app:' });


/* Few lines of code later... */


app.get('/metrics', async (req, res) => {
 res.set('Content-Type', client.register.contentType);
 res.send(await client.register.metrics());
});
```

Prom-client

Prom-client browser output

```
# HELP my_app:process_cpu_user_seconds_total Total user CPU time spent in
seconds.
# TYPE my_app:process_cpu_user_seconds_total counter
my_app:process_cpu_user_seconds_total 0.099753

# HELP my_app:process_cpu_system_seconds_total Total system CPU time spent in
seconds.
# TYPE my_app:process_cpu_system_seconds_total counter
my_app:process_cpu_system_seconds_total 0.035999

# HELP my_app:process_cpu_seconds_total Total user and system CPU time spent
in seconds.
# TYPE my_app:process_cpu_seconds_total counter
my_app:process_cpu_seconds_total 0.135752

# HELP my_app:process_start_time_seconds Start time of the process since unix
epoch in seconds.
# TYPE my_app:process_start_time_seconds gauge
my_app:process_start_time_seconds 1617192073
```

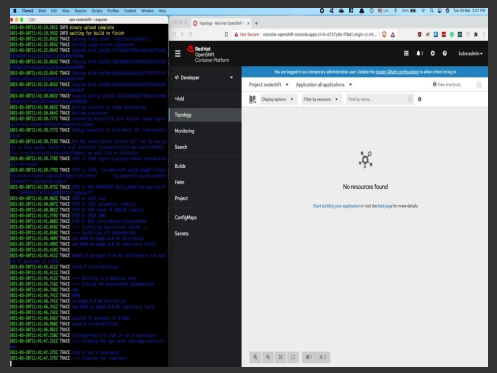
What to Collect?

RED Metrics

The RED Method defines the three key metrics you should measure for every microservice in your architecture. These metrics are part of the ***Four Golden Signals*** series defined by Google Site Reliability Engineering.

- Rate - the number of requests, per second, your services are serving.
- Errors - the number of failed requests per second.
- Duration - distributions of the amount of time each request takes.

Demo

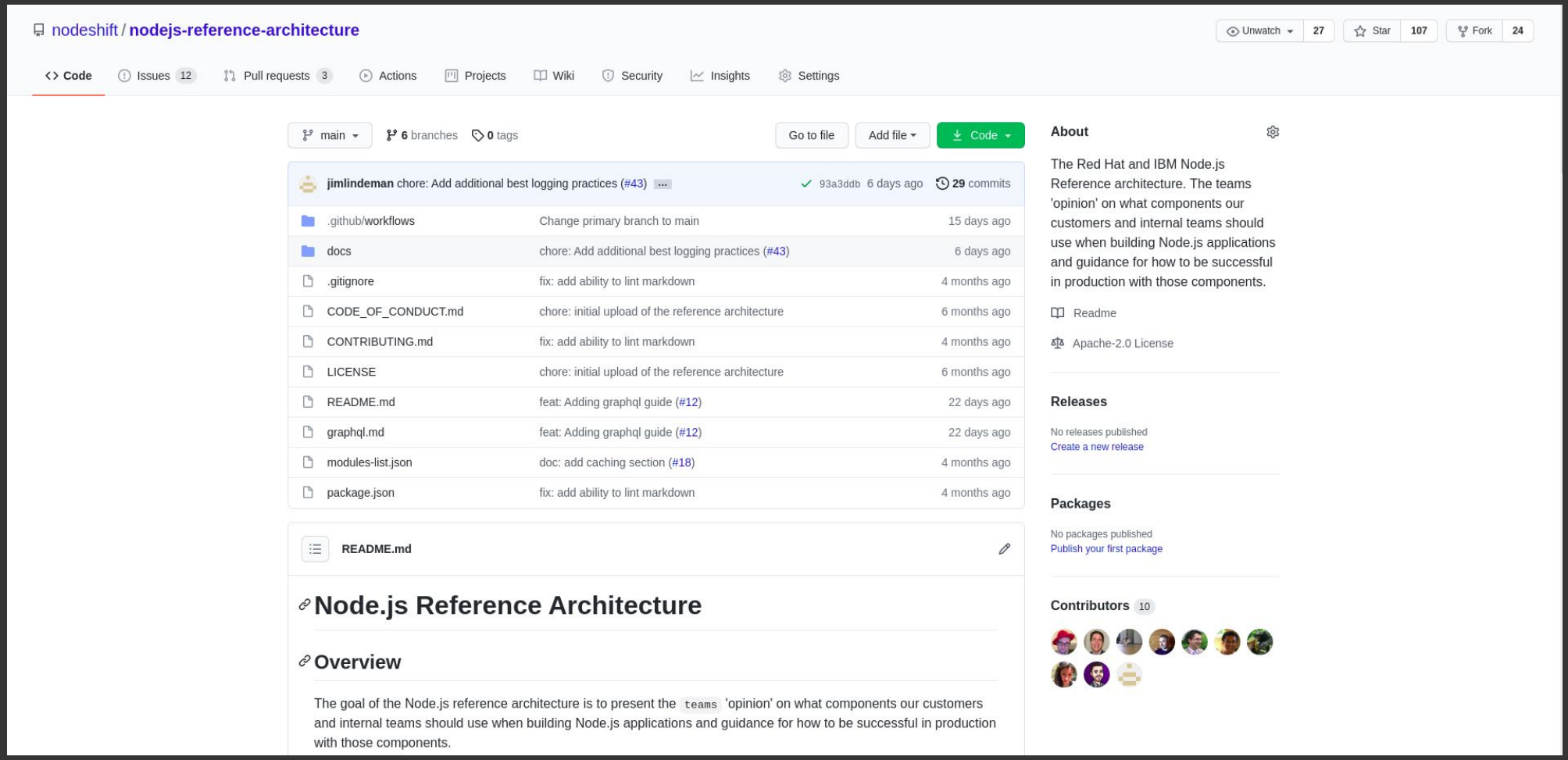

Red Hat
OpenShift
Container Platform
Developer
+Add
Topology
Monitoring
Search
Builds
Helm
Project
ConfigMaps
Secrets
No resources found

Red Hat

Service Monitor

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: nodeshift-monitor
  name: nodeshift-monitor
  namespace: default
spec:
  endpoints:
    - interval: 30s
      port: http
      scheme: http
  selector:
    matchLabels:
      project: my-app
```

Reference Architecture



https://github.com/nodeshift/nodejs-reference-architecture

Wrap Up

Monitoring Node Applications on Openshift

Prometheus

Prom-client

Reference Architecture

nodeshift.dev

Thanks


linkedin.com/company/red-hat


youtube.com/user/RedHatVideos


facebook.com/redhatinc


twitter.com/RedHat

Red Hat