NODEJS PERFORMANCE ANALYSIS

CHRISTIAN HENNIG

ScalaHacker @ inoio.de

caraboides

caraboides

DISCLAIMER

I'll show you what I've tried to solve my problem.

DISCLAIMER

I'll show you what I've tried to solve my problem.

But I failed! I did not find the solution for my problem.

DISCLAIMER

I'll show you what I've tried to solve my problem.

But I failed! I did not find the solution for my problem.

Maybe you have an idea. ;-)

OVERVIEW

OVERVIEW

- What's my problem

OVERVIEW

- What's my problem
- How to find bottlenecks with monitoring

OVERVIEW

- What's my problem
- How to find bottlenecks with monitoring
- How to get and analyze CPU profiles

OVERVIEW

- What's my problem
- How to find bottlenecks with monitoring
- How to get and analyze CPU profiles
- How to get and analyze heap dumps

OVERVIEW

- What's my problem
- How to find bottlenecks with monitoring
- How to get and analyze CPU profiles
- How to get and analyze heap dumps
- Tracing with DataDog

TASCHENHERZ

PROBLEM

PROBLEM

SERVICE RESPONSE TIME > 1SEC

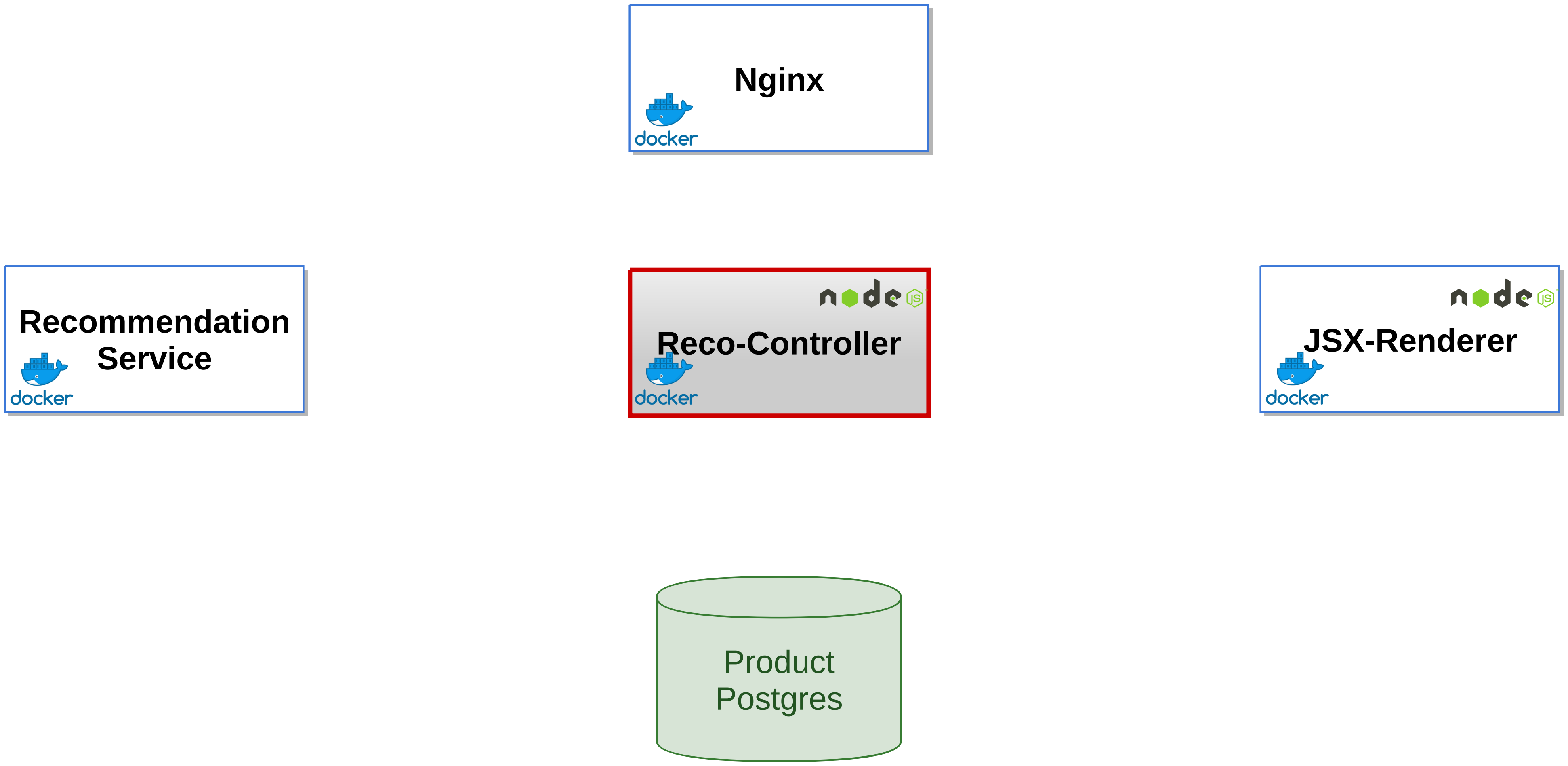
PROBLEM

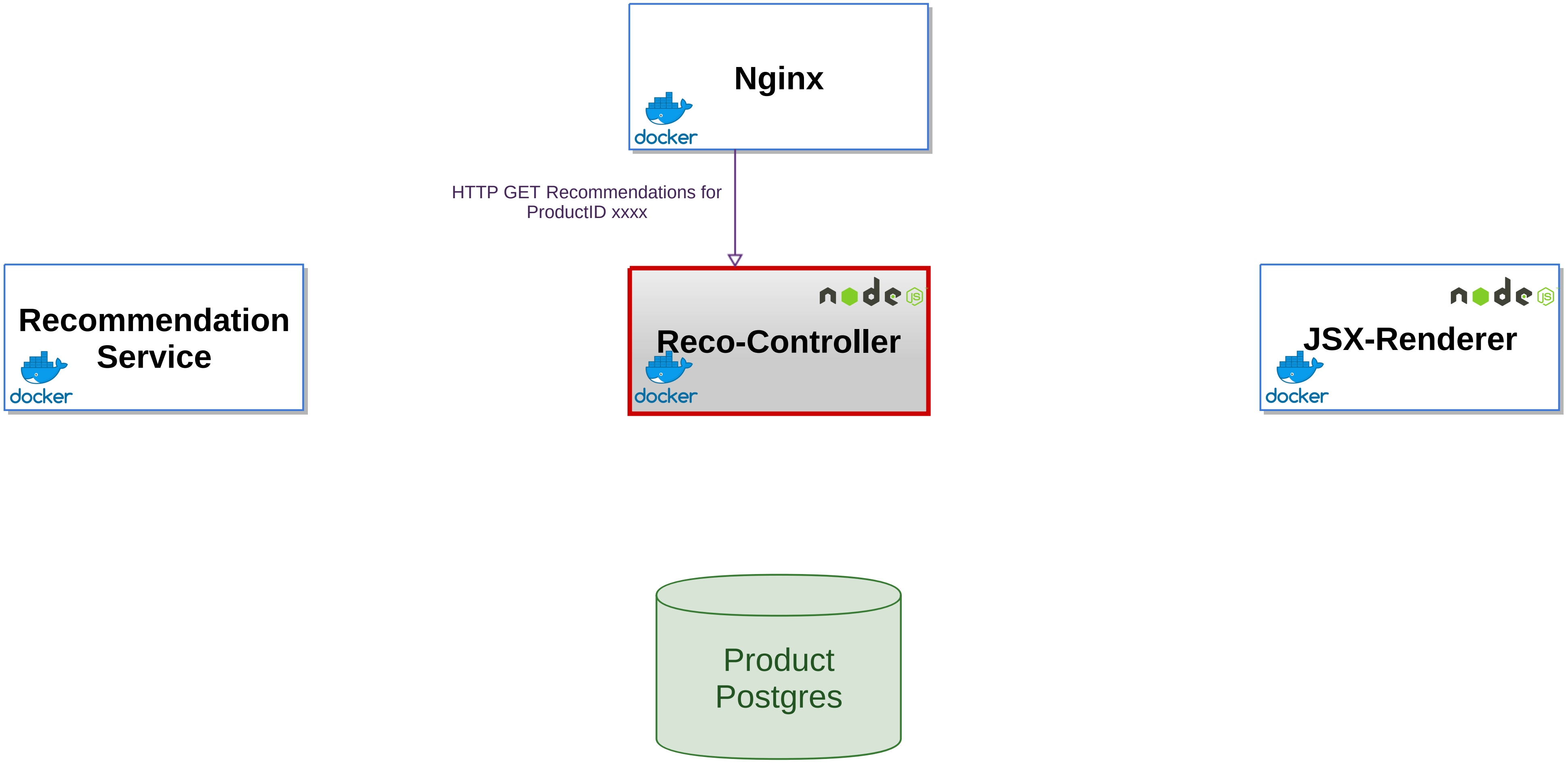
SERVICE RESPONSE TIME > 1SEC

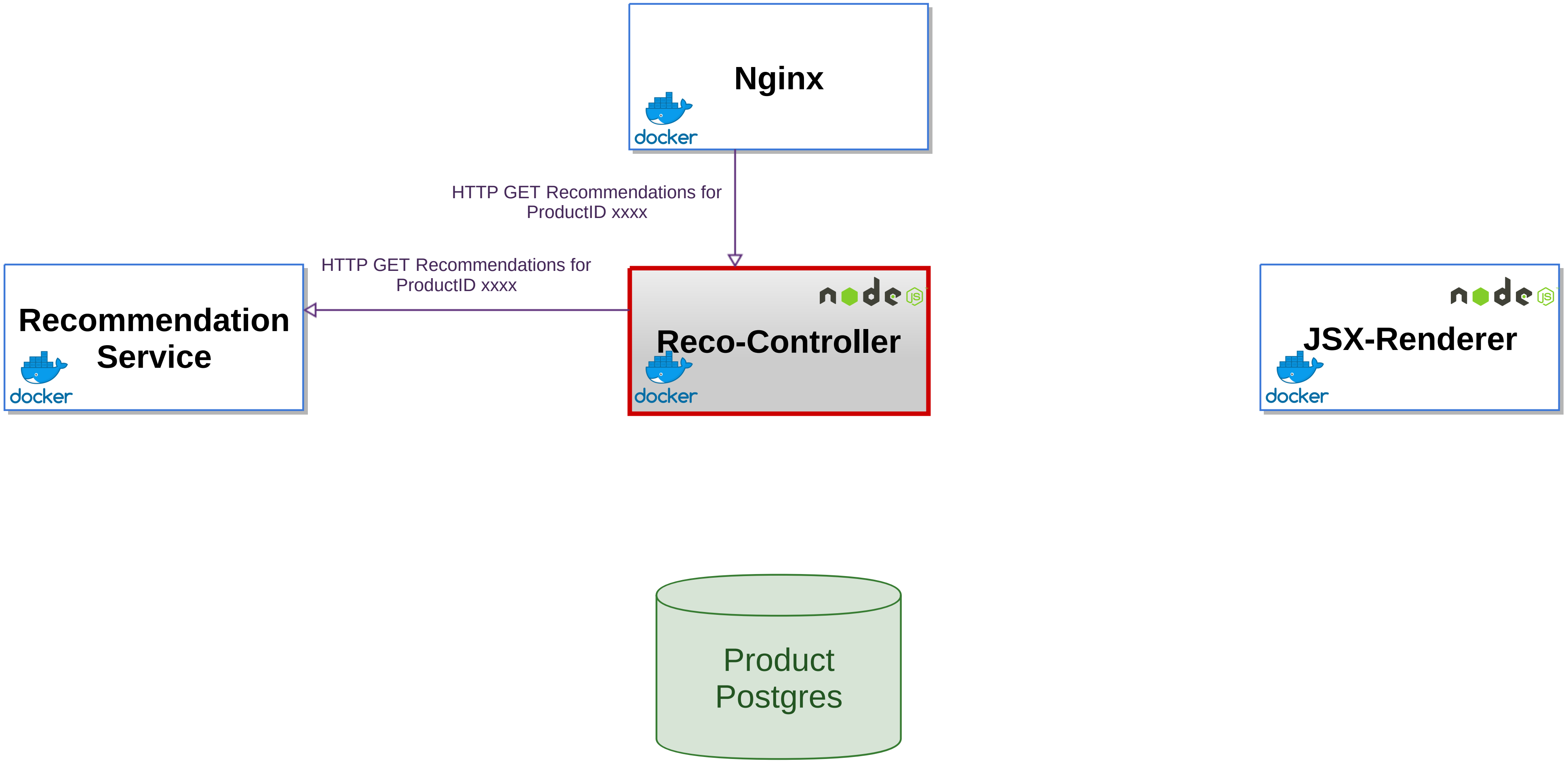
NGINX ABORTS REQUESTS AFTER 1 SEC

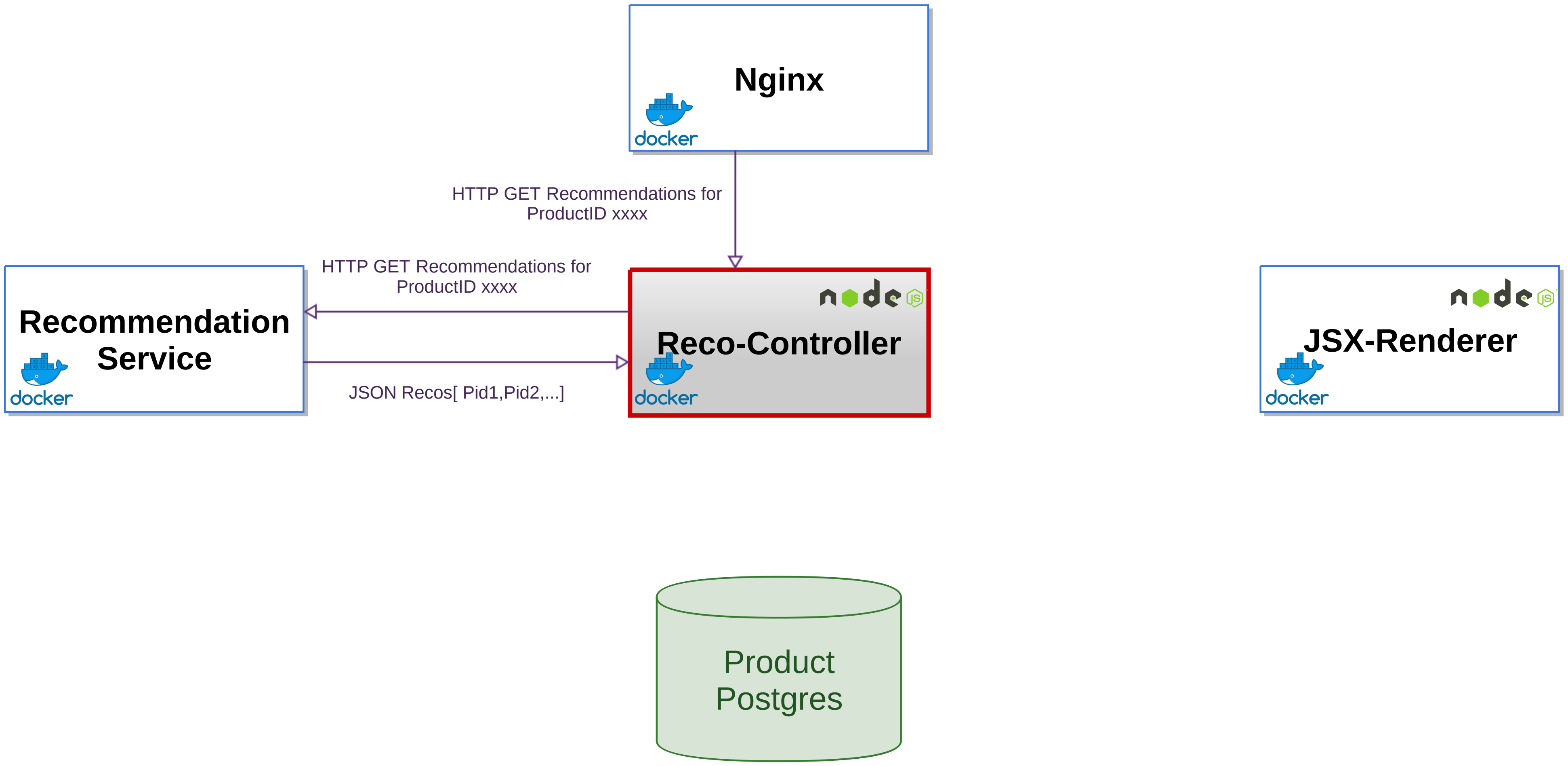
Reco-Controller

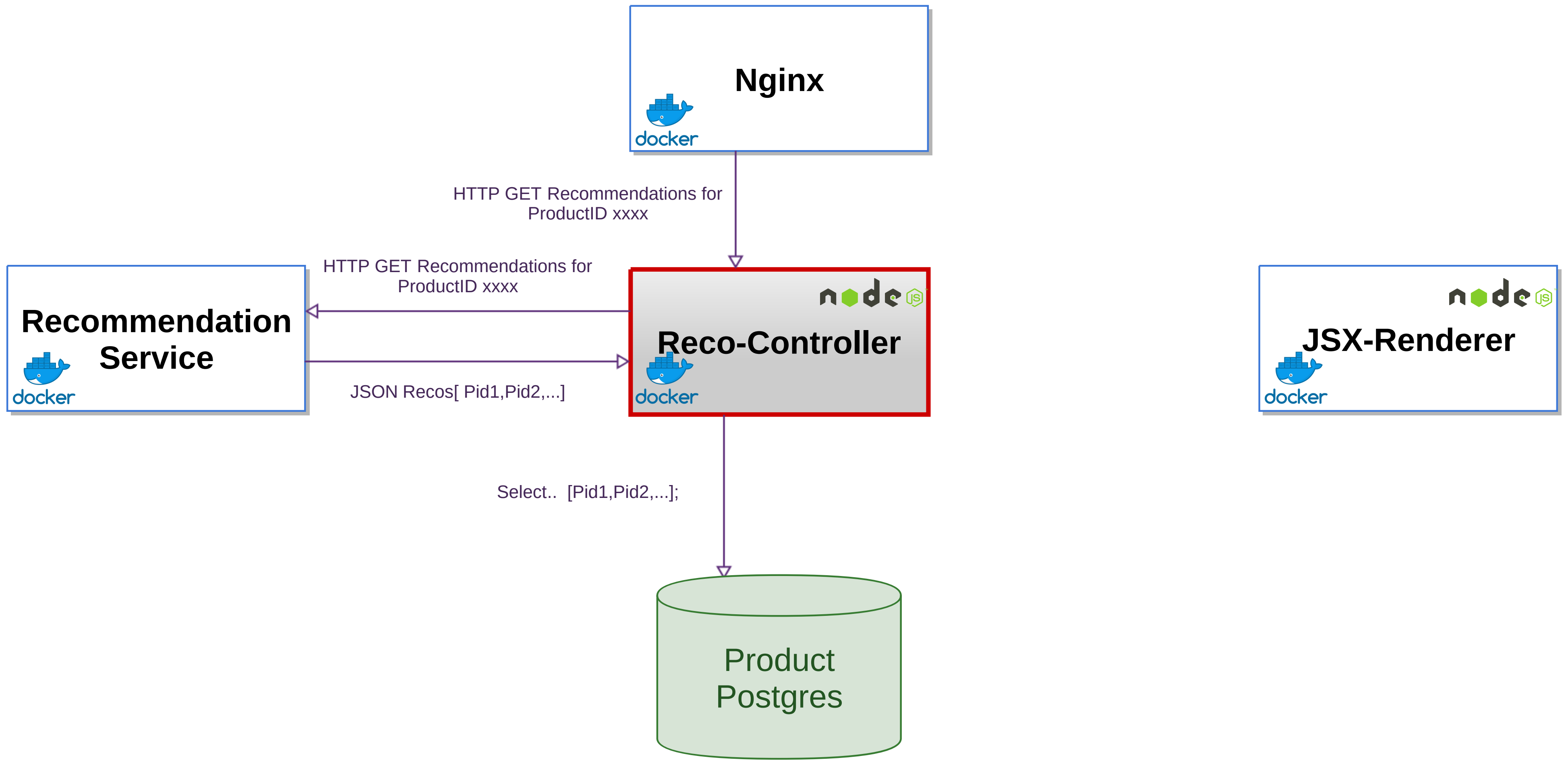
MicroServices:-)

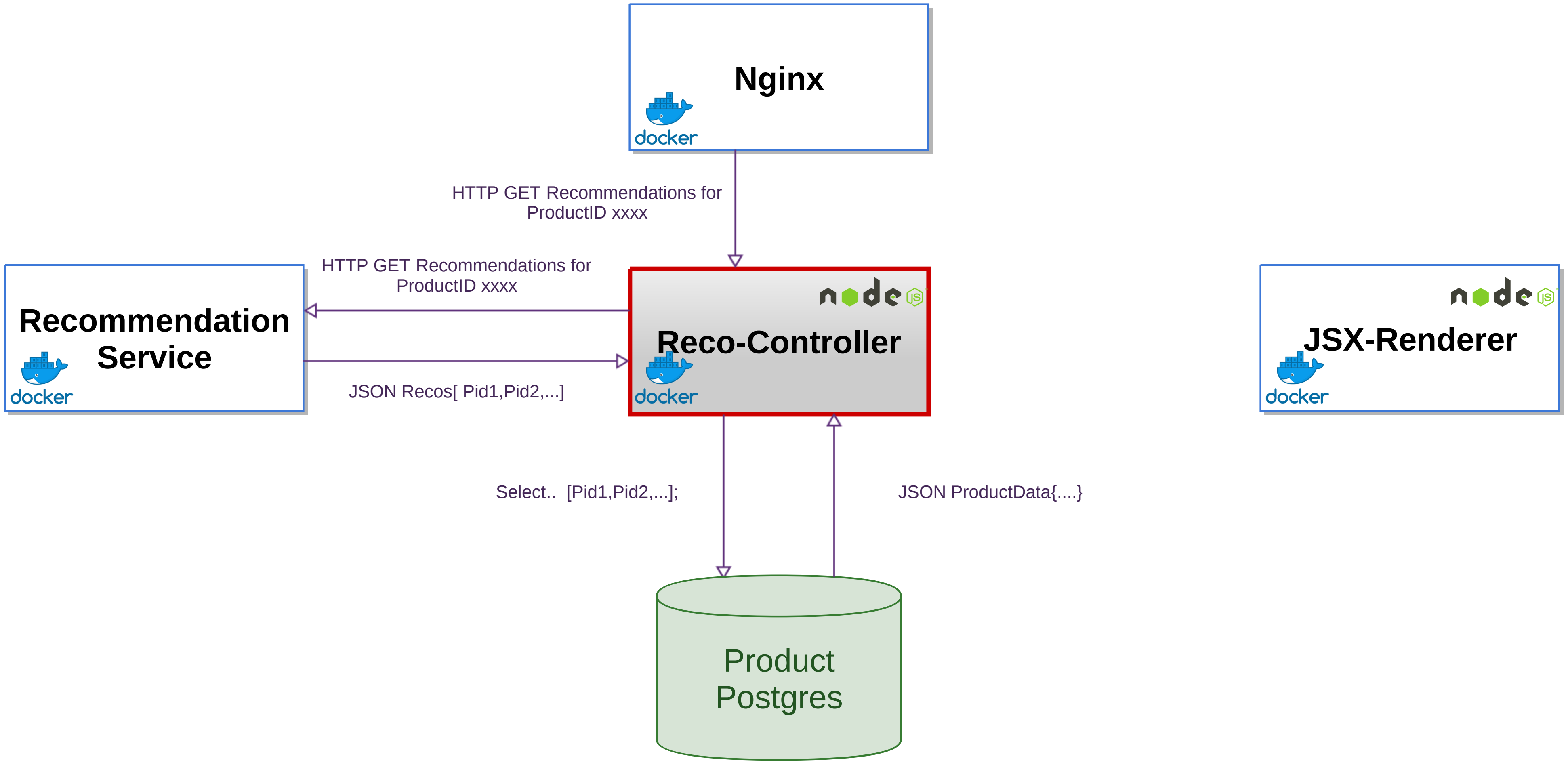

Nginx
docker
Recommendation Service
docker
node JS
Reco-Controller
docker
node JS
JSX-Renderer
docker
Product Postgres

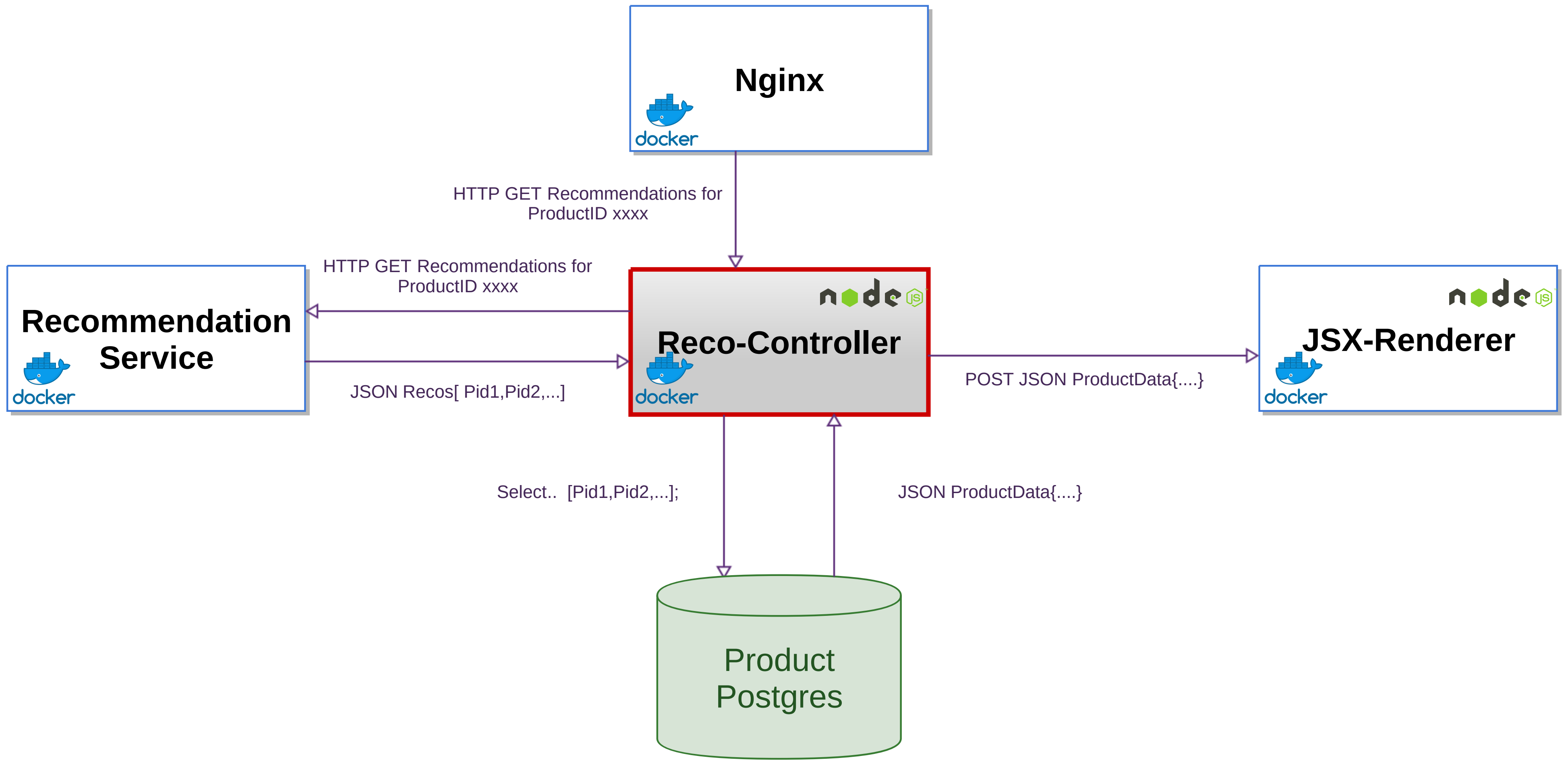

Nginx
docker
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
docker
node js
Reco-Controller
docker
node js
JSX-Renderer
docker
Product Postgres

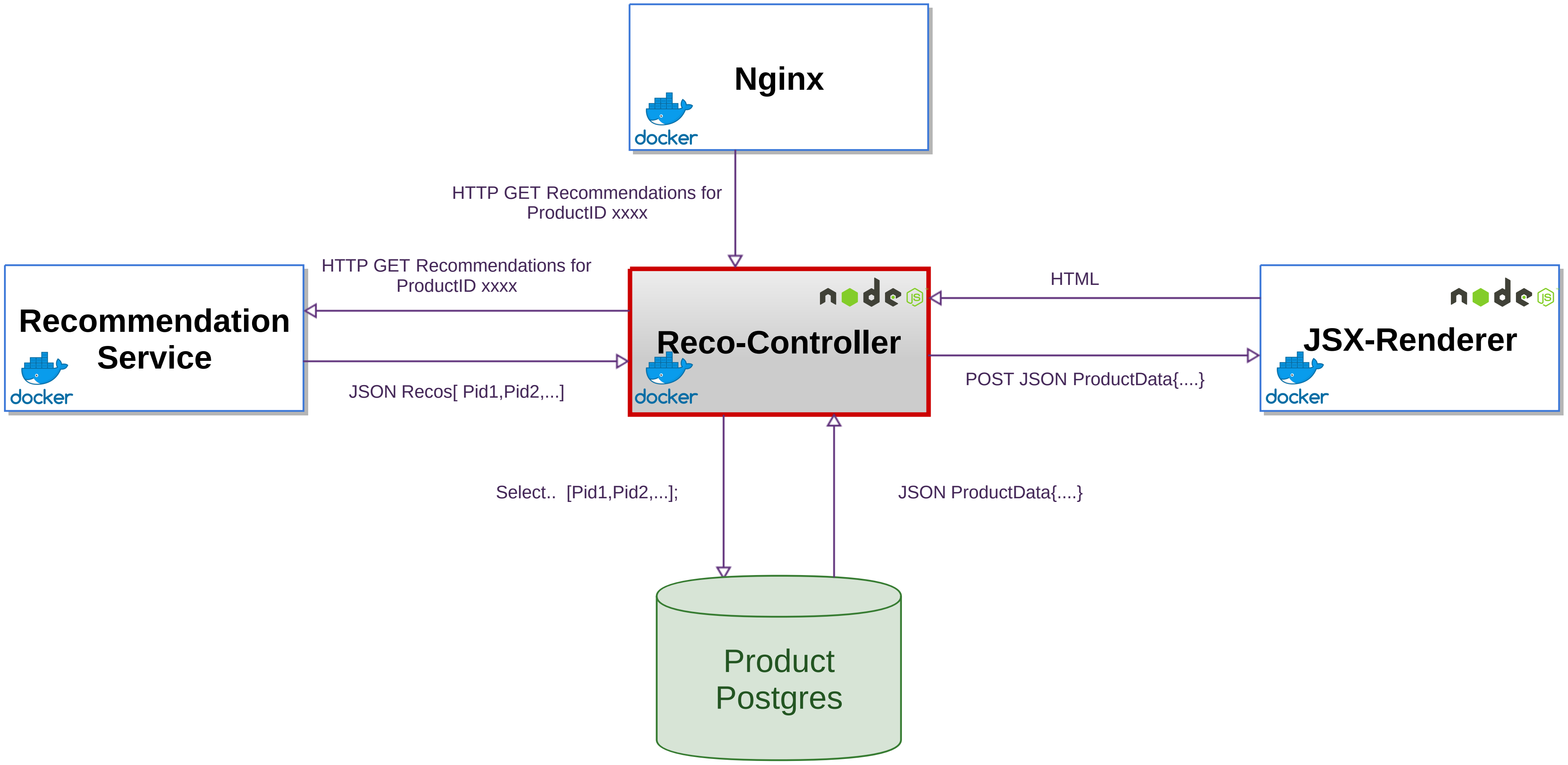

Nginx
docker
HTTP GET Recommendations for ProductID xxxx
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
docker
node JS
Reco-Controller
docker
node JS
JSX-Renderer
docker
Product Postgres

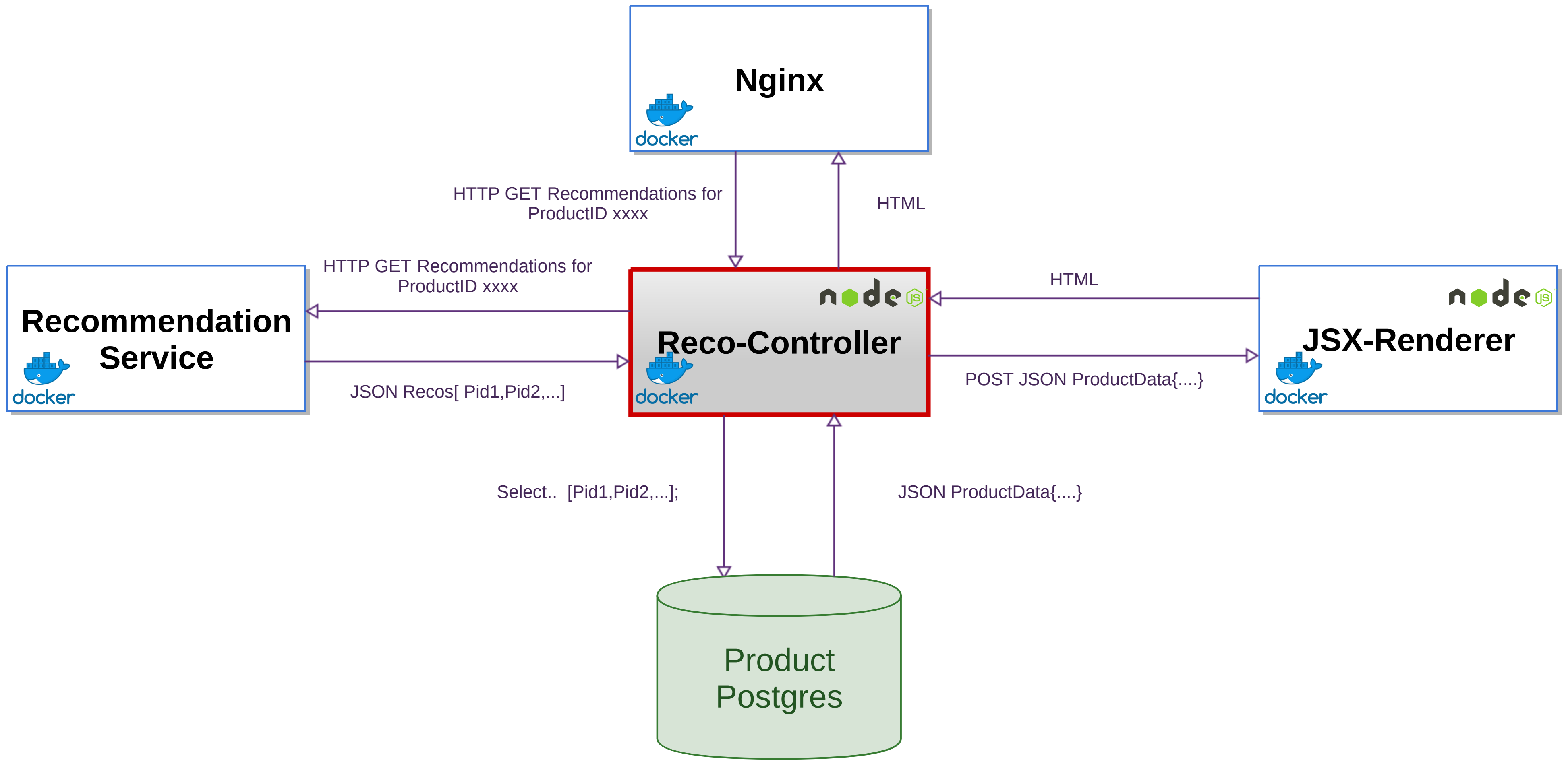

Nginx
HTTP GET Recommendations for ProductID xxxx
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
Reco-Controller
JSON Recos[Pid1,Pid2,...]
JSX-Renderer
Product Postgres
docker
node JS

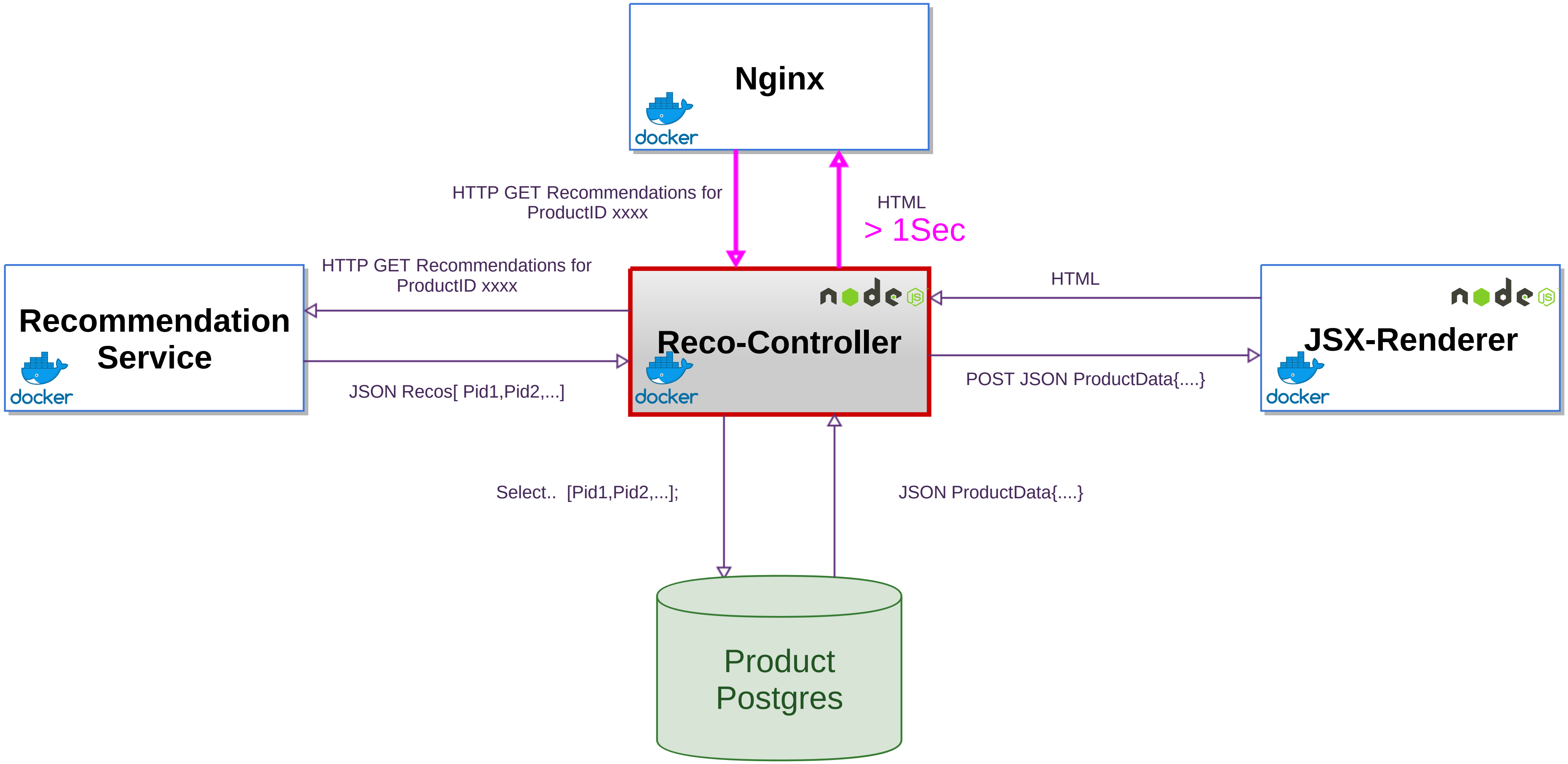

Nginx
HTTP GET Recommendations for ProductID xxxx
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
Reco-Controller
JSX-Renderer
JSON Recos[Pid1,Pid2,...]
Select.. [Pid1,Pid2,...];
Product Postgres


Nginx
docker
HTTP GET Recommendations for ProductID xxxx
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
docker
JSON Recos[Pid1,Pid2,...]
node JS
Reco-Controller
docker
node JS
JSX-Renderer
docker
Select.. [Pid1,Pid2,...];
JSON ProductData{....}
Product Postgres


Nginx
docker
HTTP GET Recommendations for ProductID xxxx
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
docker
Reco-Controller
node JS
docker
JSON Recos[Pid1,Pid2,...]
POST JSON ProductData{....}
JSX-Renderer
node JS
docker
Select.. [Pid1,Pid2,...];
JSON ProductData{....}
Product Postgres


Nginx
docker
HTTP GET Recommendations for ProductID xxxx
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
docker
JSON Recos[Pid1,Pid2,...]
Reco-Controller
node JS
docker
HTML
POST JSON ProductData{....}
JSX-Renderer
node JS
docker
Select.. [Pid1,Pid2,...];
JSON ProductData{....}
Product Postgres


Nginx
docker
HTTP GET Recommendations for ProductID xxxx
HTML
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
docker
JSON Recos[Pid1,Pid2,...]
Reco-Controller
node JS
docker
HTML
POST JSON ProductData{....}
JSX-Renderer
node JS
docker
Select.. [Pid1,Pid2,...];
JSON ProductData{....}
Product Postgres


Nginx
docker
HTTP GET Recommendations for ProductID xxxx
HTML
> 1Sec
HTTP GET Recommendations for ProductID xxxx
Recommendation Service
docker
JSON Recos[Pid1,Pid2,...]
node JS
Reco-Controller
docker
HTML
POST JSON ProductData{....}
node JS
JSX-Renderer
docker
Select.. [Pid1,Pid2,...];
JSON ProductData{....}
Product Postgres

STEP 1

STEP 1

- Steady and reproducible environment

STEP 1

- Steady and reproducible environment
- Documentation of every step: idea, change, impact

STEP 1

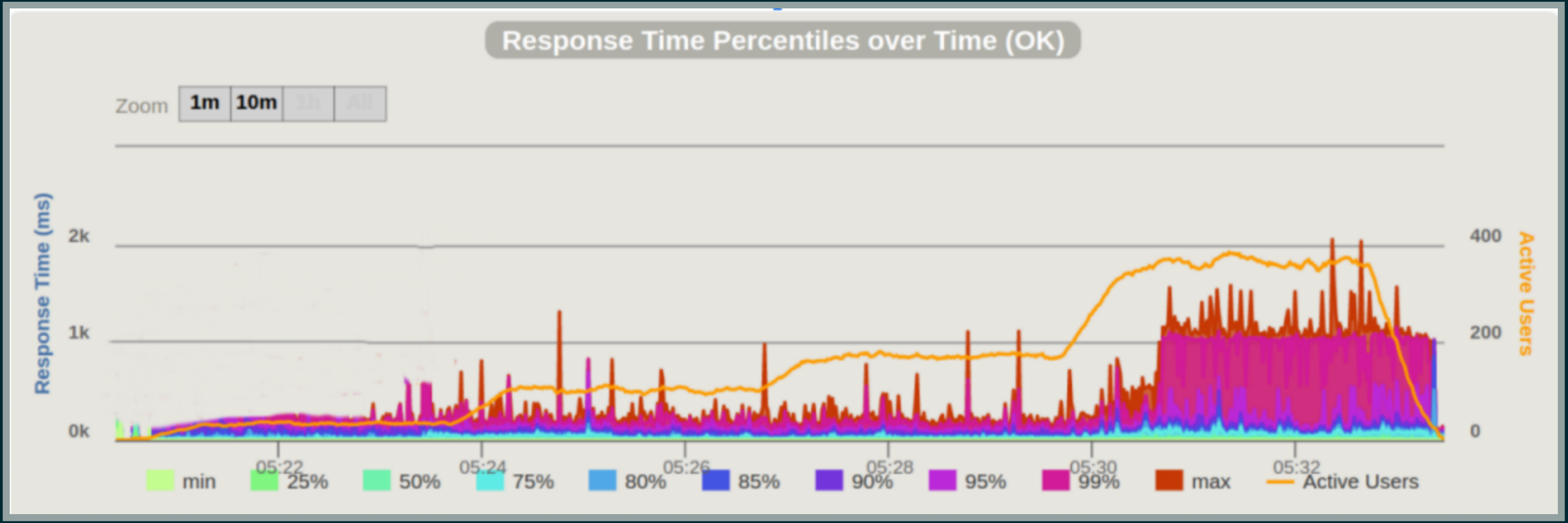
- Steady and reproducible environment
- Documentation of every step: idea, change, impact
- Periodic and automatic tests to reproduce your issue

STEP 1

- Steady and reproducible environment
- Documentation of every step: idea, change, impact
- Periodic and automatic tests to reproduce your issue
- I use Gatling

STEP 1

- Steady and reproducible environment
- Documentation of every step: idea, change, impact
- Periodic and automatic tests to reproduce your issue
- I use Gatling



STEP 2

STEP 2

- Add monitoring to your NodeJS application

STEP 2

- Add monitoring to your NodeJS application
- System which saves and displays your monitoring data

STEP 2

- Add monitoring to your NodeJS application
- System which saves and displays your monitoring data
- Prometheus + Grafana

STEP 2

- Add monitoring to your NodeJS application
- System which saves and displays your monitoring data
- Prometheus + Grafana
- Don't use average! Use percentiles like 95p to see spikes!

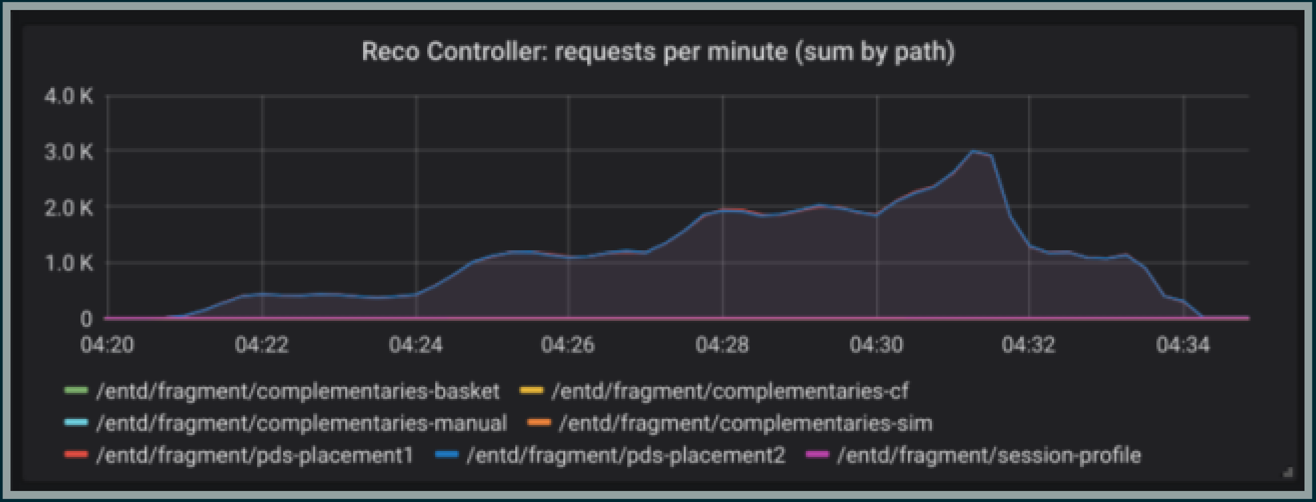
STEP 2

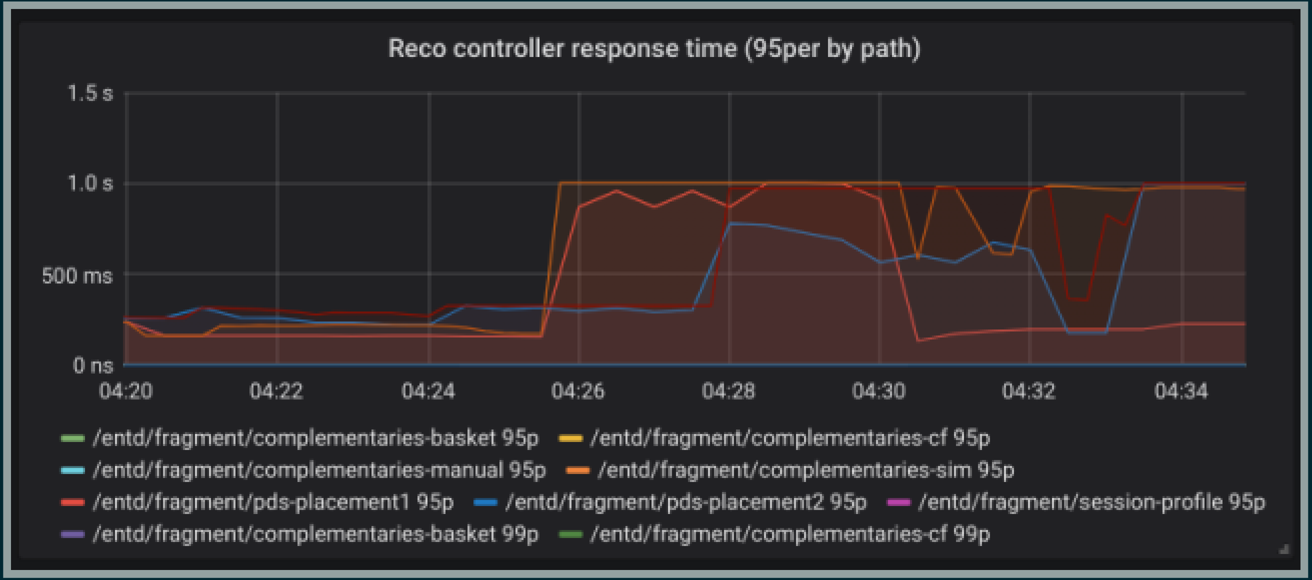
- Add monitoring to your NodeJS application
- System which saves and displays your monitoring data
- Prometheus + Grafana
- Don't use average! Use percentiles like 95p to see spikes!
- npm i --save express-prom-bundle **5.1.0**

STEP 2

- Add monitoring to your NodeJS application
- System which saves and displays your monitoring data
- Prometheus + Grafana
- Don't use average! Use percentiles like 95p to see spikes!
- npm i --save express-prom-bundle **5.1.0**

```
const metricsMiddleware = require('express-prom-bundle')({
        percentiles: [ 0.5, 0.75, 0.9, 0.95, 0.99],
        metricType: 'summary',
        maxAgeSeconds: 300,
        ageBuckets: 5
})
const app = express()
app.use(metricsMiddleware)
```


Reco Controller: requests per minute (sum by path)
4.0 K
3.0 K
2.0 K
1.0 K
0
04:20
04:22
04:24
04:26
04:28
04:30
04:32
04:34
/entd/fragment/complementaries-basket
/entd/fragment/complementaries-cf
/entd/fragment/complementaries-manual
/entd/fragment/complementaries-sim
/entd/fragment/pds-placement1
/entd/fragment/pds-placement2
/entd/fragment/session-profile


Reco controller response time (95per by path)
1.5 s
1.0 s
500 ms
0 ns
04:20
04:22
04:24
04:26
04:28
04:30
04:32
04:34
/entd/fragment/complementaries-basket 95p
/entd/fragment/complementaries-cf 95p
/entd/fragment/complementaries-manual 95p
/entd/fragment/complementaries-sim 95p
/entd/fragment/pds-placement1 95p
/entd/fragment/pds-placement2 95p
/entd/fragment/session-profile 95p
/entd/fragment/complementaries-basket 99p
/entd/fragment/complementaries-cf 99p

STEP 3

STEP 3

- Add more monitoring

STEP 3

- Add more monitoring
- Monitor critical code paths

STEP 3

- Add more monitoring
- Monitor critical code paths
- Calls to other Services

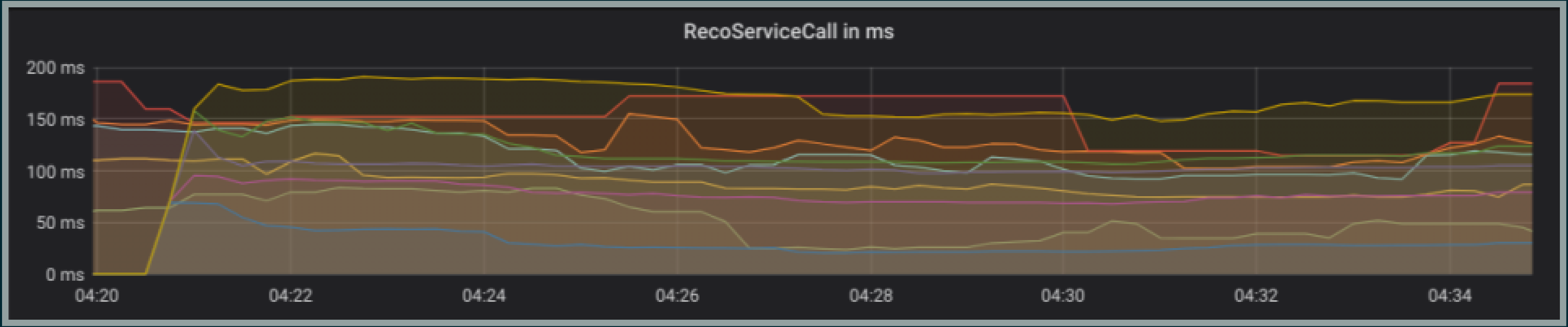
STEP 3

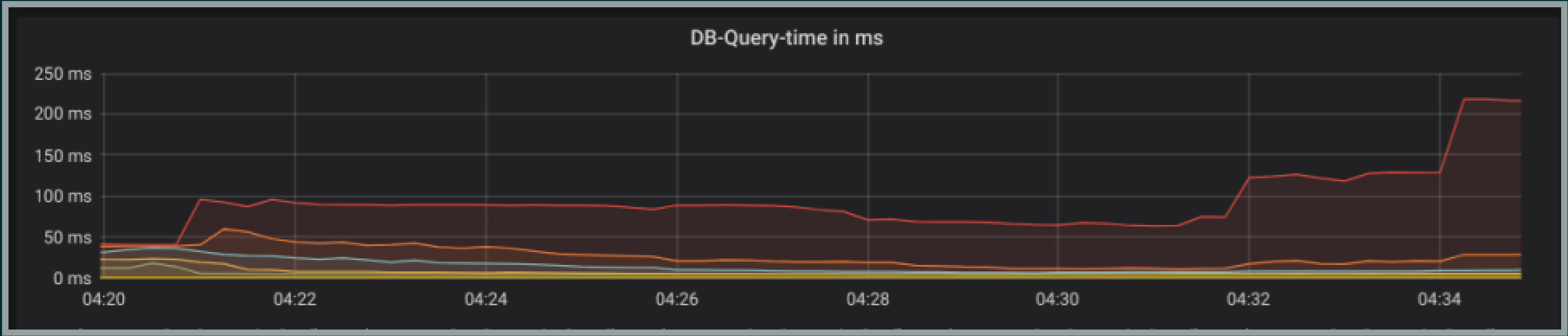
- Add more monitoring
- Monitor critical code paths
- Calls to other Services
- CPU intensive paths, e.g. json-parsing or crypto

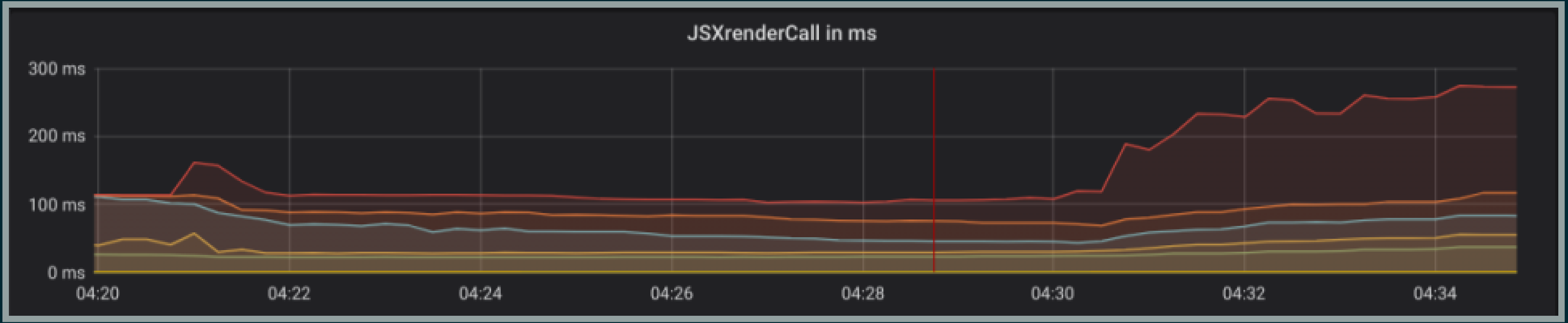
```
const promClient = require('prom-client')
const externalCallMonitor = new promClient.Summary({
        name: 'external_call_duration_ms',
        percentiles: [0.5, 0.75, 0.9, 0.95, 0.99],
        maxAgeSeconds: 300,
        ageBuckets: 5
})
```

```
const promClient = require('prom-client')
const externalCallMonitor = new promClient.Summary({
        name: 'external_call_duration_ms',
        percentiles: [0.5, 0.75, 0.9, 0.95, 0.99],
        maxAgeSeconds: 300,
        ageBuckets: 5
})
```

```
const requeststart = process.hrtime()
request({uri: "http://reco-server/reco?pid=1"}).
then((body) => {
  const requestduration = process.hrtime(requeststart)
  externalCallMonitor.observe(requestduration[1] / 1000000)
})
```


RecoServiceCall in ms
200 ms
150 ms
100 ms
50 ms
0 ms
04:20
04:22
04:24
04:26
04:28
04:30
04:32
04:34


DB-Query-time in ms
250 ms
200 ms
150 ms
100 ms
50 ms
0 ms
04:20
04:22
04:24
04:26
04:28
04:30
04:32
04:34


JSXrenderCall in ms
300 ms
200 ms
100 ms
0 ms
04:20
04:22
04:24
04:26
04:28
04:30
04:32
04:34

STEP 4

STEP 4

- Add more monitoring

STEP 4

- Add more monitoring
- Monitor the whole business code path

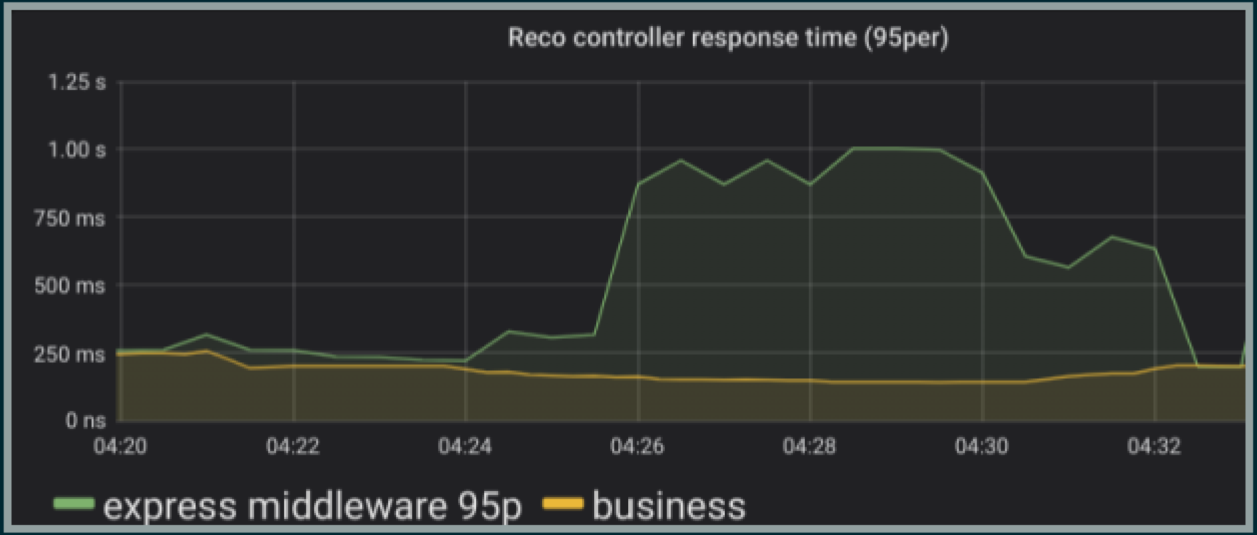
STEP 4

- Add more monitoring
- Monitor the whole business code path
- From request handling to writing the response

```
const promClient = require('prom-client')
const businessLogicMonitor = new promClient.Summary({
        name: 'business_logic_duration_ms',
        percentiles: [0.5, 0.75, 0.9, 0.95, 0.99],
        maxAgeSeconds: 300,
        ageBuckets: 5
})
```

```
const promClient = require('prom-client')
const businessLogicMonitor = new promClient.Summary({
        name: 'business_logic_duration_ms',
        percentiles: [0.5, 0.75, 0.9, 0.95, 0.99],
        maxAgeSeconds: 300,
        ageBuckets: 5
})
```

```
router.get('/recos', (req, res) => {
  const businessLogicStart = process.hrtime()
  businessLogic(req.body)  // Transform Data, Call 3 external Services
  .then((html) => {
        const businessDuration = process.hrtime(businessLogicStart)
        businessLogicMonitor.observe(businessDuration[1] / 1000000)
        res.send(html)
  })
})
```


Reco controller response time (95per)
1.25 s
1.00 s
750 ms
500 ms
250 ms
0 ns
04:20
04:22
04:24
04:26
04:28
04:30
04:32
express middleware 95p
business

STEP 5

STEP 5

- CPU Profiling

STEP 5

- CPU Profiling
- node --prof app.js | Was not an option

STEP 5

- CPU Profiling
- node --prof app.js | Was not an option
- npm i --save v8-profiler-node8

```
const express = require('express')
const router = express.Router()
const profiler = require("v8-profiler-node8")

router.get('/cpuprofile', (req, res) => {
  const id = Date.now() + ".profile"
  profiler.startProfiling(id)
          // stop profiling in n seconds and exit
        setTimeout(() => {
        res.set('Content-Type', 'application/json-home')
        res.json(profiler.stopProfiling(id))
        profiler.deleteAllProfiles()
}, 10000)
})
module.exports = router
```

Chrome analysis

STEP 6

STEP 6

- Heap Dumps

STEP 6

- Heap Dumps
- npm i --save heapdump

```javascript
const express = require('express')
const router = express.Router()
const heapdump = require('heapdump')
const fileSystem = require('fs')
router.get('/heapdump', (req, response) => {
        heapdump.writeSnapshot(function(err, filename) {
        var stat = fileSystem.statSync(filename);
        response.writeHead(200, {
                'Content-Type': 'application/octet-stream',
                'Content-Length': stat.size
        });
        var readStream = fileSystem.createReadStream(filename);
        readStream.pipe(response);
        });
})
module.exports = router
```

Chrome analysis

STEP 7

STEP 7

- Add more monitoring

STEP 7

- Add more monitoring
- Monitor GarbageCollection and Internals of nodes

STEP 7

- Add more monitoring
- Monitor GarbageCollection and Internals of nodes
- npm i --save prometheus-gc-stats

```javascript
const prometheus = require('prom-client')
prometheus.collectDefaultMetrics()

const gcStats = require('prometheus-gc-stats')
const startGcStats = gcStats(prometheus.register)
startGcStats();
```

Demo Board

STEP 8

STEP 8

- Tracing with DataDog

STEP 8

- Tracing with DataDog
- npm i --save dd-trace

STEP 8

- Tracing with DataDog
- npm i --save dd-trace

```
const tracer = require('dd-trace').init()
// enable and configure postgresql integration
tracer.use('pg', {
  service: 'pg-cluster'
})
// enable and configure express integration
tracer.use('express', {
  service: 'express-cluster'
})
```

DD-DEMO

MY NEXT STEPS

MY NEXT STEPS

- Local setup to reproduce the issue: docker-compose + blockade

MY NEXT STEPS

- Local setup to reproduce the issue: docker-compose + blockade
- Take a deeper look into tcp.connect, [http|tcp]_keep_alive

MY NEXT STEPS

- Local setup to reproduce the issue: docker-compose + blockade
- Take a deeper look into tcp.connect, [http|tcp]_keep_alive
- Rewrite in scala: service issue or environment issue?

QUESTIONS ?

IDEAS ?

Slides: https://github.com/inoio/node-perf-talk

@caraboides