Otimizando uma aplicação Node.js
Deep dive

Gustavo Oliveira

Antes de tudo, dois avisos:

1. A apresentação terá muito código, mas o objetivo é entender a ideia;

2. As vezes vou fazer alguma pergunta, parar a apresentação e olhar no chat se alguém respondeu. Com isso, dá tempo de todo mundo assimilar o código.

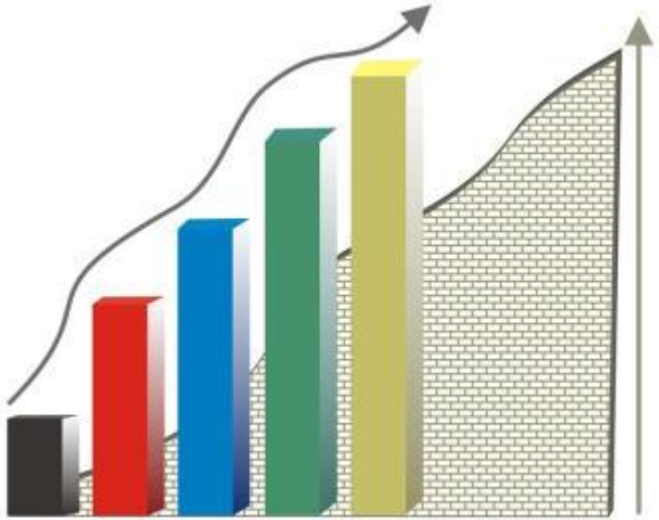
Sumário

1. Breve contexto;
2. Threads em Node.js;
3. Processos em Node.js;
4. Mais Threads;
5. Como abstrair tudo isso em produção.

Auto Scaling

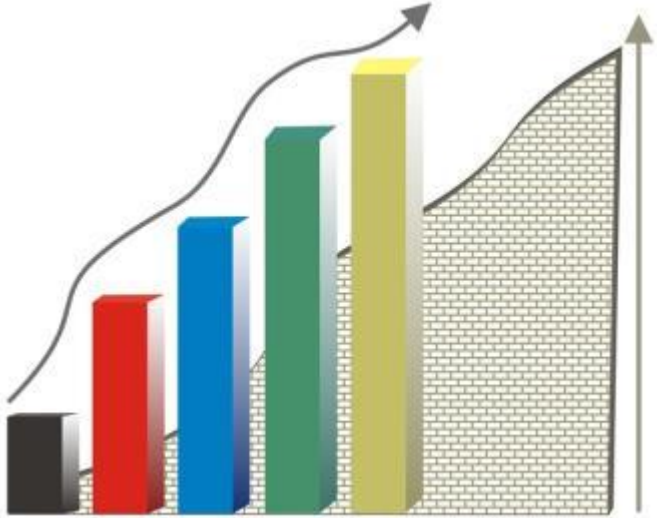
Escalabilidade vertical x horizontal

Escalabilidade vertical



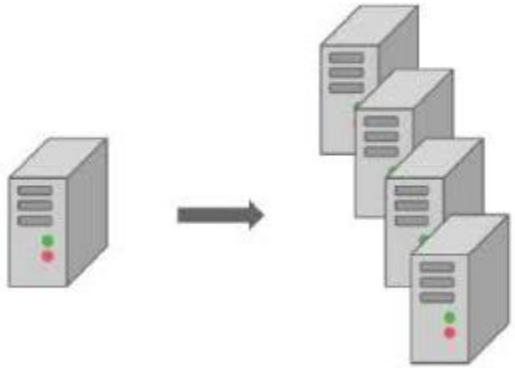
Escalabilidade vertical

Servidor com recursos poderosos seriam muito caros



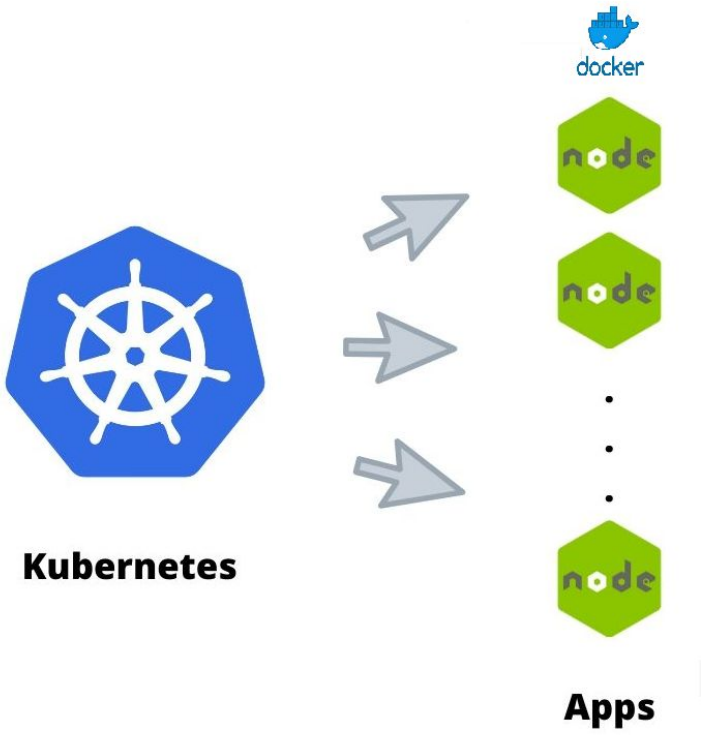
Escalabilidade horizontal

Não daria tempo de subir máquinas tão rápido em um servidor tradicional



Escalabilidade horizontal

Escalabilidade horizontal



Como depender menos de infraestrutura?

Serverless?

Serverless?

Case mais famoso é o AWS Lambda

https://thedevconf.com/palestrante/gustavo-oliveira

Se não tivermos nada “sofisticado” desse tipo, temos que trabalhar com o que temos! E agora?

Se a vida te der um limão, faça uma limonada!

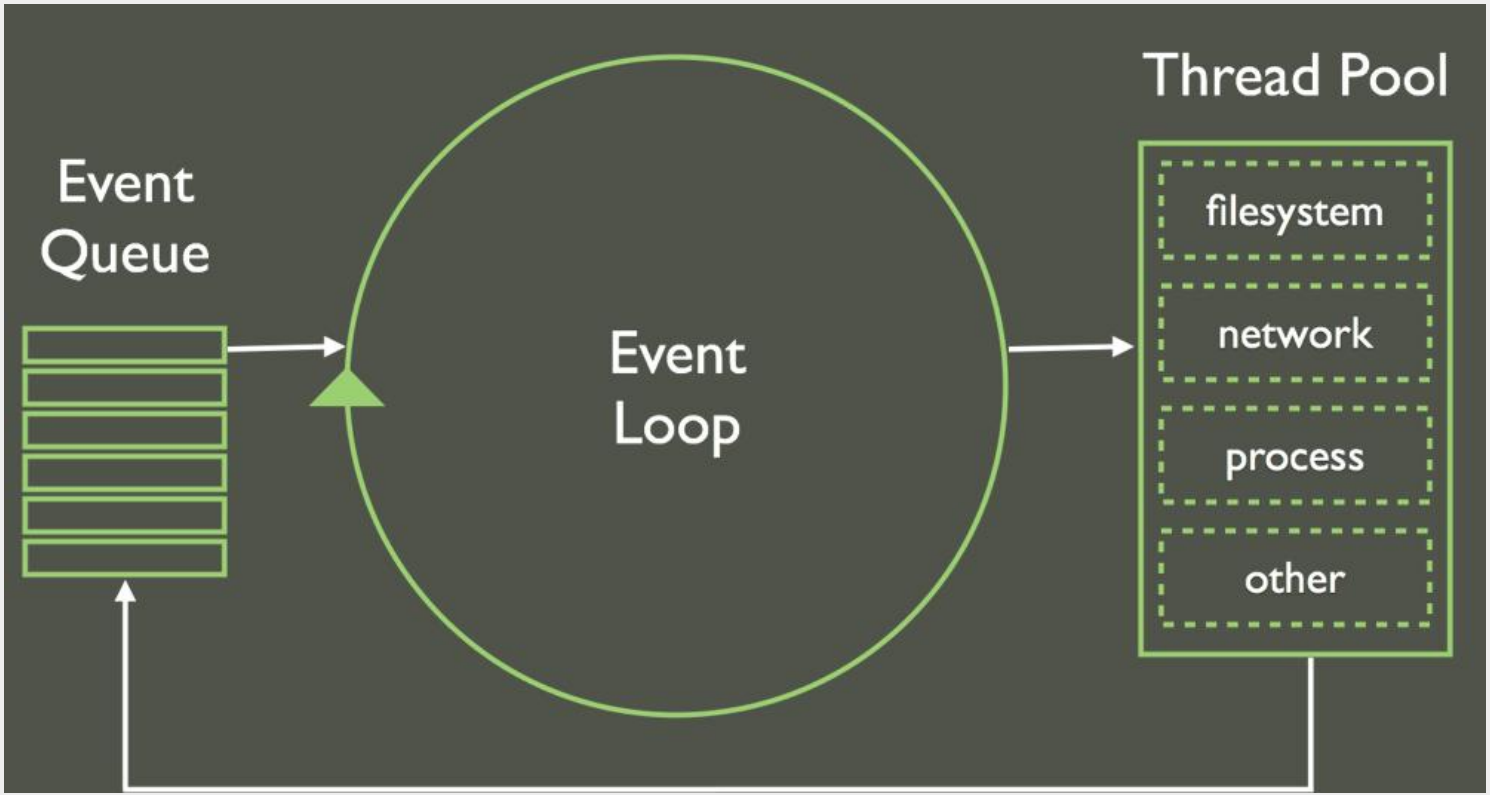
Clarice Lispector

Quantas vezes alguém já te disse: ‘Não bloqueie a main-thread do JavaScript’?

Node.js usa dois tipos de threads

- A main thread manipulada pelo *event loop*;
- Várias threads auxiliares no *worker pool*.

Event loop e thread pool



Thread pool

Thread pool

```
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', (err, derivedKey) => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
```

Thread pool

```
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
```

```
gustavo@gustavo:~/tdc$ node index.js
Hash:  476
```

Thread pool

```
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
logHashTime();
```

```
gustavo@gustavo:~/tdc$ node index.js
Hash:  487
Hash:  491
```

Thread pool

```
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

```
gustavo@gustavo:~/tdc$ node index.js
Hash:  510
Hash:  516
Hash:  534
Hash:  538
```

Thread pool

```
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

```
gustavo@gustavo:~/tdc$ node index.js
Hash:  520
Hash:  526
Hash:  526
Hash:  526
Hash:  1011
```

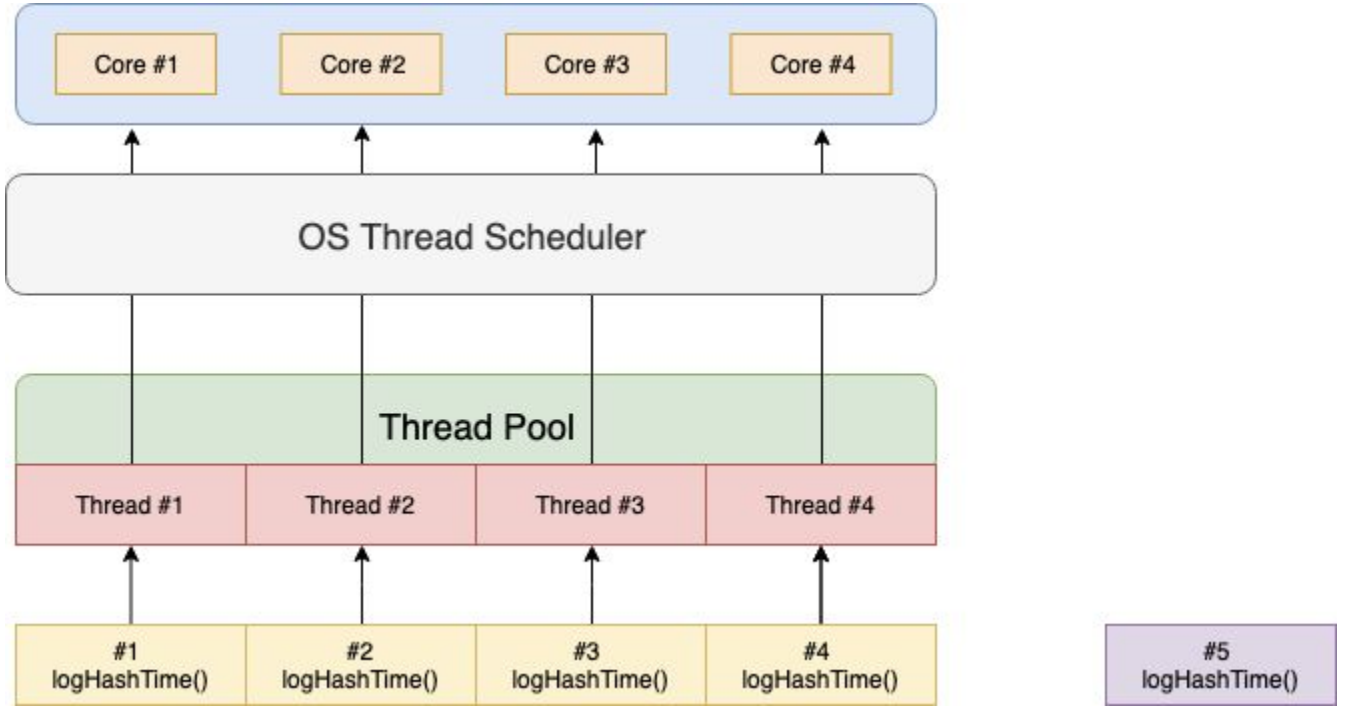
Thread pool

```
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

```
gustavo@gustavo:~/tdc$ node index.js
Hash:  520
Hash:  526
Hash:  526
Hash:  526
Hash:  1011
```

Por que esse demorou mais?

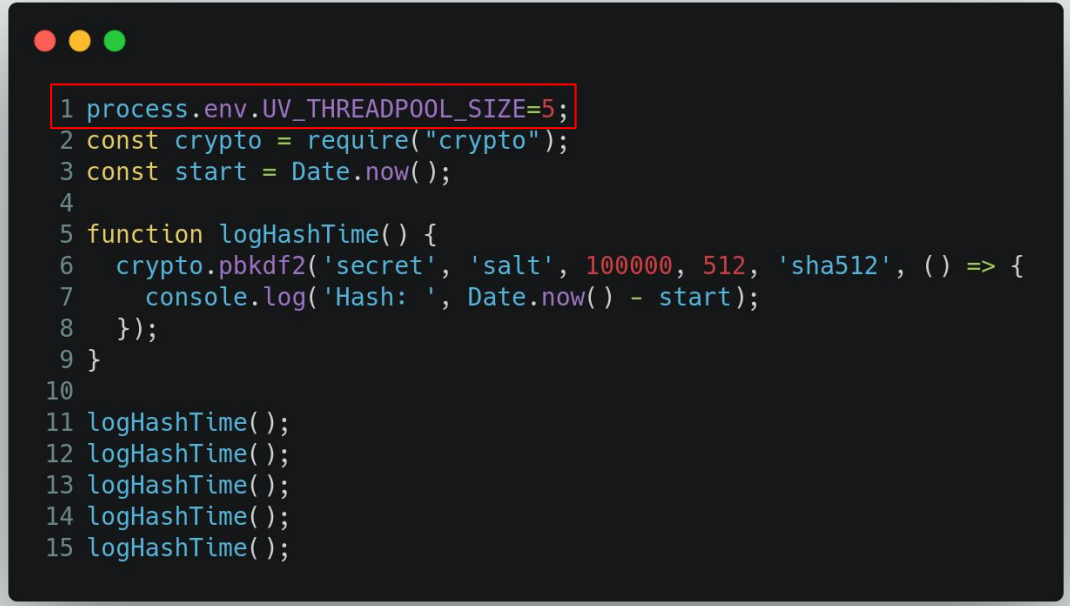
Thread pool


Core #1
Core #2
Core #3
Core #4
OS Thread Scheduler
Thread Pool
Thread #1
Thread #2
Thread #3
Thread #4
#1 logHashTime()
#2 logHashTime()
#3 logHashTime()
#4 logHashTime()
#5 logHashTime()

Thread pool

```
process.env.UV_THREADPOOL_SIZE=5;
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

Thread pool


process.env.UV_THREADPOOL_SIZE=5;
const crypto = require("crypto");
const start = Date.now();
function logHashTime() {
crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
console.log('Hash: ', Date.now() - start);
});
}
logHashTime();
logHashTime();
logHashTime();
logHashTime();
logHashTime();

Thread pool

```node
process.env.UV_THREADPOOL_SIZE=5;
const crypto = require("crypto");
const start = Date.now();

function logHashTime() {
  crypto.pbkdf2('secret', 'salt', 100000, 512, 'sha512', () => {
    console.log('Hash: ', Date.now() - start);
  });
}

logHashTime();
logHashTime();
logHashTime();
logHashTime();
logHashTime();
```

```
gustavo@gustavo:~/tdc$ node index.js
Hash:  519
Hash:  524
Hash:  524
Hash:  769
Hash:  769
```

Com isso em mente, vamos adiante...

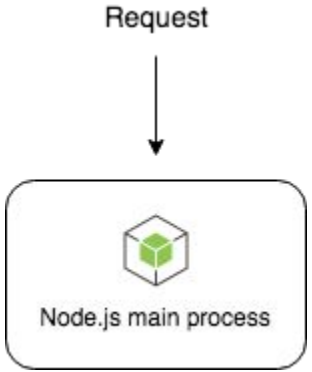
Temos basicamente dois módulos que performam aplicações com uso intensivo de CPU:

- cluster - process based
  - Por baixo dos panos usa um módulo chamado child_process
- worker_threads - thread based
  - Solução elegante para o problema de concorrência do JavaScript
  - Ele não apresenta *features* de linguagens *multi-threading*, em vez disso, provê concorrência por permitir que a aplicação use múltiplos *workers* isolados

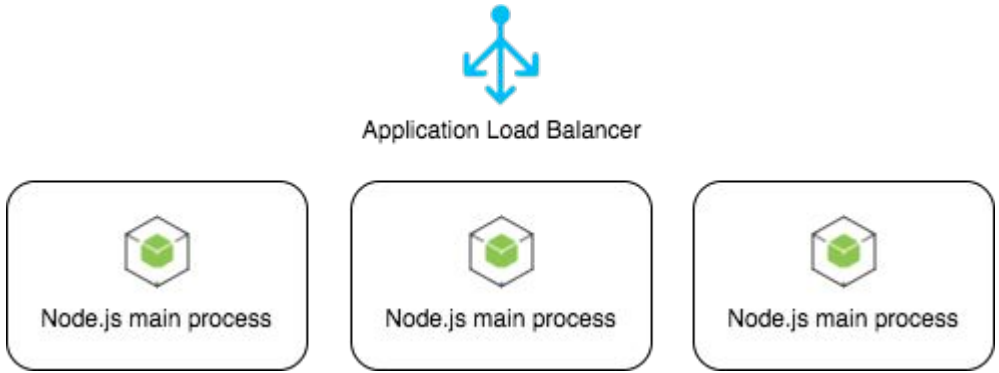
Vamos analisar primeiro o modo cluster...

Cenário sem cluster

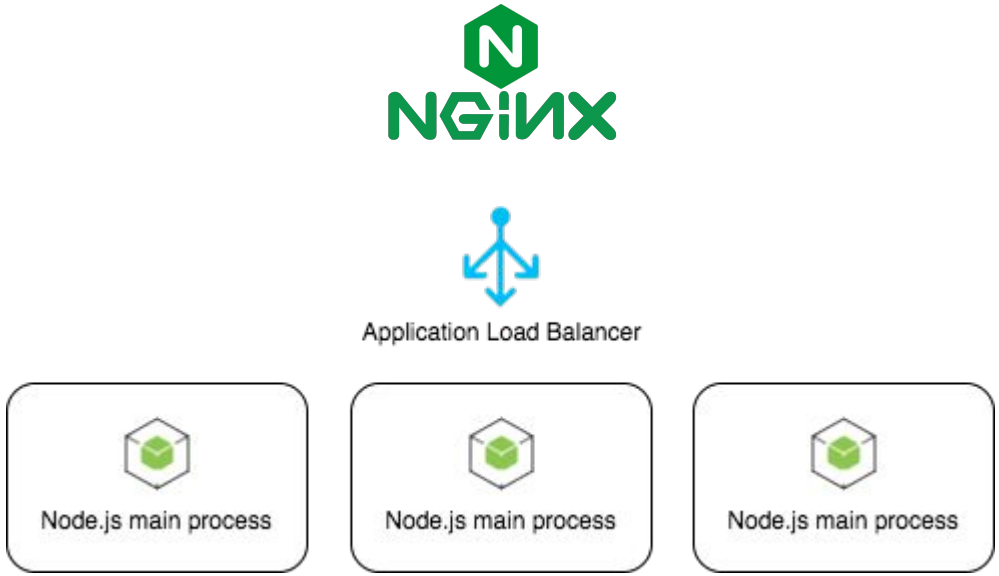
Aplicação Web Node.js simples



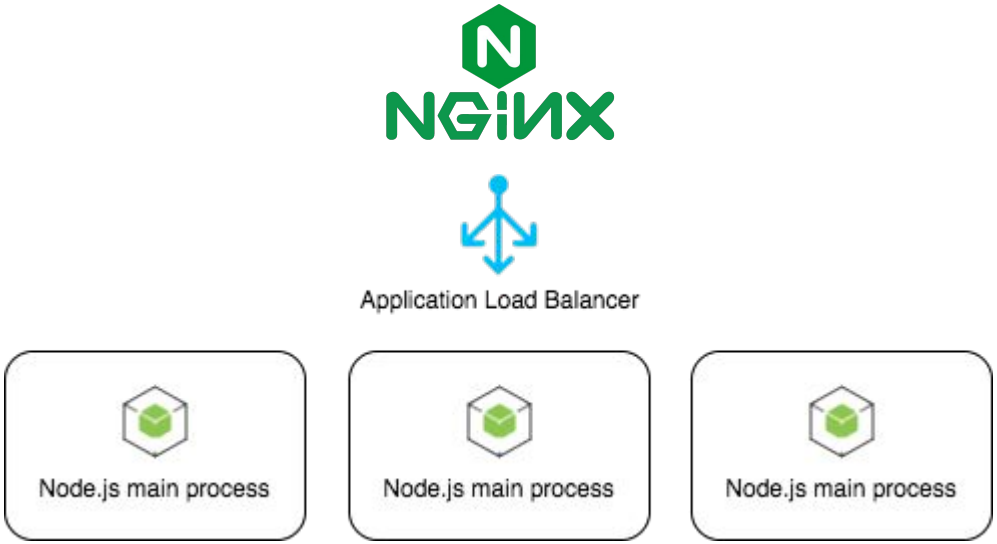
Caso a aplicação estivesse escalada horizontalmente



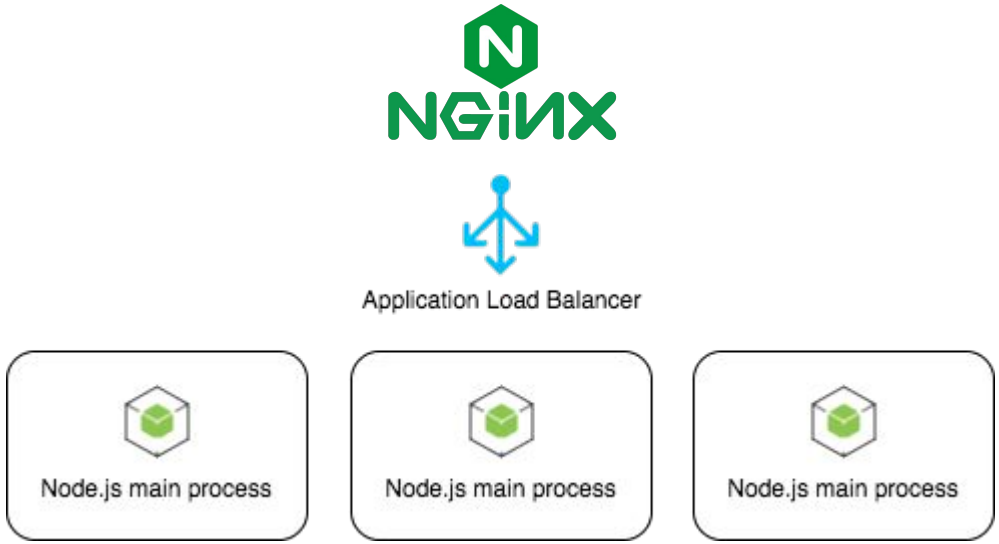
Caso a aplicação estivesse escalada horizontalmente



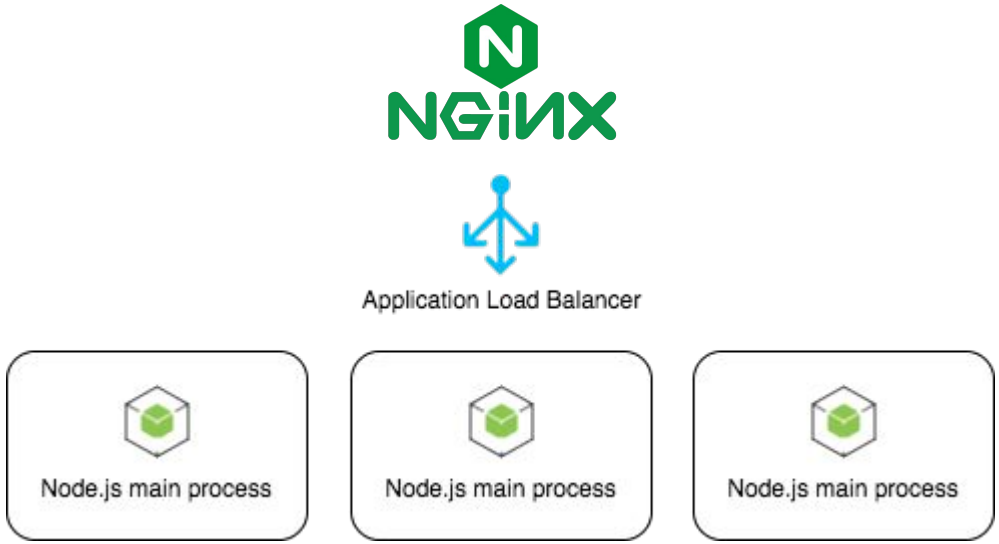
Caso a aplicação estivesse escalada horizontalmente e com **1** núcleo por máquina



Caso a aplicação estivesse escalada horizontalmente e com **2** núcleos por máquina

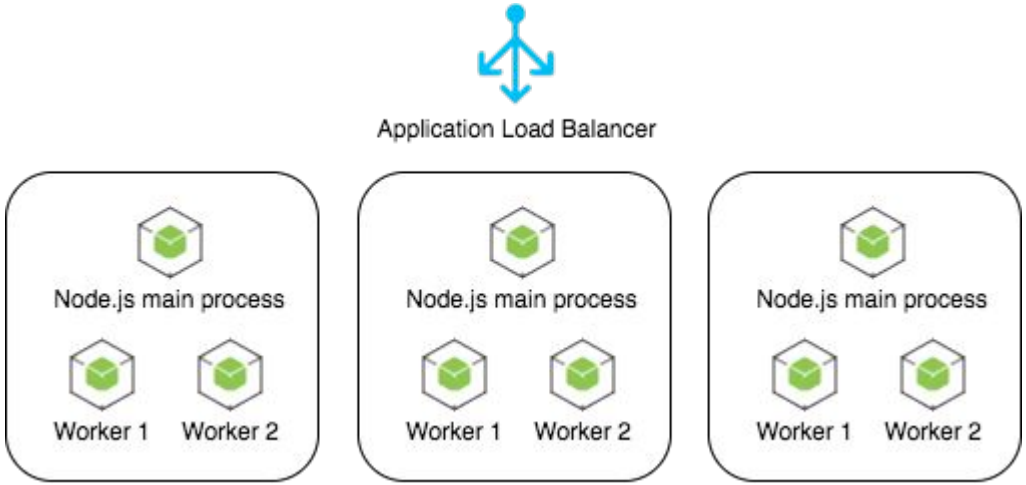


Caso a aplicação estivesse escalada horizontalmente e com **n** núcleos



Cenário com cluster

Cluster com 2 núcleos por máquina



Daí surgem algumas dúvidas...

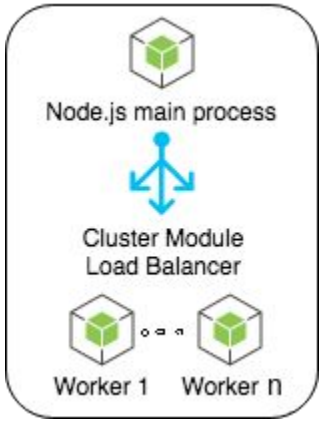
Dúvidas pertinentes

- Como os requests são processados pelos workers?

Dúvidas pertinentes

- Como os requests são processados pelos workers?
- Quem decide quais requests serão processados por quais workers?

O módulo de cluster faz todo esse trabalho!



Vamos ver na prática!

Utilização Básica - O código original

```
const express = require('express');
const setRoutes = require('./route');

const app = express();

setRoutes(app)

app.server.listen('8000');
```

Utilização Básica - O código original

```
const express = require('express');
const setRoutes = require('./route');

const app = express();

setRoutes(app)

app.server.listen('8000');
```

Utilização Básica - O código original

```
const express = require('express');

const app = express();

app.get('/', (req, res) => {
  res.status(200).send(`Hello, I am master!`);
})

app.server.listen('8000');
```

Utilização Básica - O código transformado

Vamos dissecar o código...

```
const express = require('express');
const cluster = require('cluster');
const numCores = require('os').cpus().length;

const app = express();

if (cluster.isMaster) {
  setupWorkerProcesses();
} else {
  setUpExpress(8000);
}

const setupWorkerProcesses = () => {
  const workers = [];

  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    workers.push(worker);
  }
};

const setUpExpress = (port) => {
  app.get('/', (req, res) => {
    res.status(200).send(`Hello, worker ${cluster.worker.id} responding\n`);
  })

  app.listen(port, () => {
    console.log(`Listening on port ${port} for Process Id ${process.pid} `);
  });
};
```

Utilização Básica - O código transformado

```
const express = require('express');
const cluster = require('cluster');
const numCores = require('os').cpus().length;

const app = express();

if (cluster.isMaster) {
  setupWorkerProcesses();
} else {
  setUpExpress(8000);
}

const setupWorkerProcesses = () => {
  const workers = [];

  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    workers.push(worker);
  }
};

const setUpExpress = (port) => {
  app.get('/', (req, res) => {
    res.status(200).send(`Hello, worker ${cluster.worker.id} responding\n`);
  })

  app.listen(port, () => {
    console.log(`Listening on port ${port} for Process Id ${process.pid} `);
  });
};
```

```
const express = require('express');
const cluster = require('cluster');
const numCores = require('os').cpus().length;

const app = express();
```

Utilização Básica - O código

```
if (cluster.isMaster) {
  setupWorkerProcesses();
} else {
  setUpExpress(8000);
}

const setupWorkerProcesses = () => {
  const workers = [];

  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    workers.push(worker);
  }
};

const setUpExpress = (port) => {
  app.get('/', (req, res) => {
    res.status(200).send(`Hello, worker ${cluster.worker.id} responding\n`);
  })

  app.listen(port, () => {
    console.log(`Listening on port ${port} for Process Id ${process.pid} `);
  });
};
```

Utilização Básica - O código

```
if (cluster.isMaster) {
  setupWorkerProcesses();
} else {
  setUpExpress(8000);
}

const setupWorkerProcesses = () => {
  const workers = [];

  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    workers.push(worker);
  }
};

const setUpExpress = (port) => {
  app.get('/', (req, res) => {
    res.status(200).send(`Hello, worker ${cluster.worker.id} responding\n`);
  })

  app.listen(port, () => {
    console.log(`Listening on port ${port} for Process Id ${process.pid} `);
  });
};
```

Utilização Básica - O código

```
if (cluster.isMaster) {
  setupWorkerProcesses();
} else {
  setUpExpress(8000);
}

const setupWorkerProcesses = () => {
  const workers = [];

  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    workers.push(worker);
  }
};

const setUpExpress = (port) => {
  app.get('/', (req, res) => {
    res.status(200).send(`Hello, worker ${cluster.worker.id} responding\n`);
  })

  app.listen(port, () => {
    console.log(`Listening on port ${port} for Process Id ${process.pid} `);
  });
};
```

Utilização Básica - O código

```
if (cluster.isMaster) {
  setupWorkerProcesses();
} else {
  setUpExpress(8000);
}

const setupWorkerProcesses = () => {
  const workers = [];

  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    workers.push(worker);
  }
};

const setUpExpress = (port) => {
  app.get('/', (req, res) => {
    res.status(200).send(`Hello, worker ${cluster.worker.id} responding\n`);
  })

  app.listen(port, () => {
    console.log(`Listening on port ${port} for Process Id ${process.pid} `);
  });
};
```

Utilização Básica - Rodando

```
clustering -> node app.js
Master is running with Process Id 18140, setting up 8 workers
Listening on port 8000 for Process Id 18147
Listening on port 8000 for Process Id 18148
Listening on port 8000 for Process Id 18154
Listening on port 8000 for Process Id 18155
Listening on port 8000 for Process Id 18162
Listening on port 8000 for Process Id 18172
Listening on port 8000 for Process Id 18179
Listening on port 8000 for Process Id 18187
```

Utilização Básica - O código

```
if (cluster.isMaster) {
  setupWorkerProcesses();
} else {
  setUpExpress(8000);
}

const setupWorkerProcesses = () => {
  const workers = [];

  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    workers.push(worker);
  }
};

const setUpExpress = (port) => {
  app.get('/', (req, res) => {
    res.status(200).send(`Hello, worker ${cluster.worker.id} responding\n`);
  })

  app.listen(port, () => {
    console.log(`Listening on port ${port} for Process Id ${process.pid} `);
  });
};
```

Utilização Básica - Fazendo um request

```
clustering -> curl localhost:8000
Hello, worker 1 responding
```

Utilização Básica - Fazendo vários requests

```
clustering -> curl localhost:8000
Hello, worker 1 responding
clustering -> curl localhost:8000
Hello, worker 2 responding
clustering -> curl localhost:8000
Hello, worker 7 responding
clustering -> curl localhost:8000
Hello, worker 6 responding
clustering -> curl localhost:8000
Hello, worker 4 responding
clustering -> curl localhost:8000
Hello, worker 5 responding
clustering -> curl localhost:8000
Hello, worker 3 responding
clustering -> curl localhost:8000
Hello, worker 8 responding
clustering -> curl localhost:8000
Hello, worker 1 responding
clustering -> curl localhost:8000
Hello, worker 2 responding
clustering -> curl localhost:8000
Hello, worker 7 responding
```

Utilização Básica - Fazendo vários requests

```
clustering -> curl localhost:8000
Hello, worker 1 responding
clustering -> curl localhost:8000
Hello, worker 2 responding
clustering -> curl localhost:8000
Hello, worker 7 responding
clustering -> curl localhost:8000
Hello, worker 6 responding
clustering -> curl localhost:8000
Hello, worker 4 responding
clustering -> curl localhost:8000
Hello, worker 5 responding
clustering -> curl localhost:8000
Hello, worker 3 responding
clustering -> curl localhost:8000
Hello, worker 8 responding
clustering -> curl localhost:8000
Hello, worker 1 responding
clustering -> curl localhost:8000
Hello, worker 2 responding
clustering -> curl localhost:8000
Hello, worker 7 responding
```

Agora que a aplicação está realmente operando em modo cluster, vamos explorar algumas coisas interessantes que podem ser feitas...

Worker parou de responder

```
cluster.on('exit', (worker, code, signal) => {
  console.log(`Worker ${worker.process.pid} died\
    with code: ${code}, and signal: ${signal}`);
}
```

Worker parou de responder - Reiniciando

```
cluster.on('exit', (worker, code, signal) ⇒ {
  console.log(`Worker ${worker.process.pid} died\
    with code: ${code}, and signal: ${signal}`);

  console.log('Starting a new worker');
  cluster.fork();
});
```

Trecho do código, até agora

```
const setupWorkerProcesses = () ⇒ {
  console.log(`Master is running for Process Id ${process.pid}, cluster setting up
    ${numCores} workers`);

  for (let i = 0; i < numCores; i++) {
    let worker = cluster.fork();
    workers.push(worker);
  }

  cluster.on('exit', function (worker, code, signal) {
    console.log(`Worker ${worker.process.pid} died with code: ${code} and signal:
      ${signal}.`);

    console.log('Starting a new worker');

    let worker = cluster.fork();
    workers.push(worker);
  });
};
```

Trecho do código, até agora

```
const setupWorkerProcesses = () => {
  console.log(`Master is running for Process Id ${process.pid}, cluster setting up
    ${numCores} workers`);

  for (let i = 0; i < numCores; i++) {
    let worker = cluster.fork();
    workers.push(worker);
  }

  cluster.on('exit', function (worker, code, signal) {
    console.log(`Worker ${worker.process.pid} died with code: ${code} and signal:
      ${signal}.`);

    console.log('Starting a new worker');

    let worker = cluster.fork();
    workers.push(worker);
  });
};
```

Trecho do código, até agora

```
const setupWorkerProcesses = () => {
  console.log(`Master is running for Process Id ${process.pid}, cluster setting up
    ${numCores} workers`);

  for (let i = 0; i < numCores; i++) {
    let worker = cluster.fork();
    workers.push(worker);
  }

  cluster.on('exit', function (worker, code, signal) {
    console.log(`Worker ${worker.process.pid} died with code: ${code} and signal:
      ${signal}.`);

    console.log('Starting a new worker');

    let worker = cluster.fork();
    workers.push(worker);
  });
};
```

Testando - Matando e reiniciando worker

```
clustering -> node app.js
Master is running for Process Id 3718, ...
Listening on port 8000 for Process Id 3728
Listening on port 8000 for Process Id 3735
Listening on port 8000 for Process Id 3729
Listening on port 8000 for Process Id 3736
Listening on port 8000 for Process Id 3743
Listening on port 8000 for Process Id 3761
Listening on port 8000 for Process Id 3750
Listening on port 8000 for Process Id 3762
```

Testando - Matando e reiniciando worker

```
clustering -> node app.js
Master is running for Process Id 3718, ...
Listening on port 8000 for Process Id 3728
Listening on port 8000 for Process Id 3735
Listening on port 8000 for Process Id 3729
Listening on port 8000 for Process Id 3736
Listening on port 8000 for Process Id 3743
Listening on port 8000 for Process Id 3761
Listening on port 8000 for Process Id 3750
Listening on port 8000 for Process Id 3762
```

Testando - Matando e reiniciando worker

```
clustering -> node app.js
Master is running for Process Id 3718, ...
Listening on port 8000 for Process Id 3728
Listening on port 8000 for Process Id 3735
Listening on port 8000 for Process Id 3729
Listening on port 8000 for Process Id 3736
Listening on port 8000 for Process Id 3743
Listening on port 8000 for Process Id 3761
Listening on port 8000 for Process Id 3750
Listening on port 8000 for Process Id 3762
```

```
clustering -> kill 3762
```

Testando - Matando e reiniciando worker

```
clustering -> node app.js
Master is running for Process Id 3718, ...
Listening on port 8000 for Process Id 3728
Listening on port 8000 for Process Id 3735
Listening on port 8000 for Process Id 3729
Listening on port 8000 for Process Id 3736
Listening on port 8000 for Process Id 3743
Listening on port 8000 for Process Id 3761
Listening on port 8000 for Process Id 3750
Listening on port 8000 for Process Id 3762
```

```
clustering -> kill 3762
```

```
clustering -> node app.js
Master is running for Process Id 3718, ...
Listening on port 8000 for Process Id 3728
Listening on port 8000 for Process Id 3735
Listening on port 8000 for Process Id 3729
Listening on port 8000 for Process Id 3736
Listening on port 8000 for Process Id 3743
Listening on port 8000 for Process Id 3761
Listening on port 8000 for Process Id 3750
Listening on port 8000 for Process Id 3762

Worker 3762 died with code: null and\
  signal: SIGTERM
Starting a new worker
Listening on port 8000 for Process Id 3792
```

Comunicação entre processo principal e seus filhos

Comunicação entre processo principal e seus filhos

O processo principal envia uma mensagem a um worker

```
worker.send('message');
```

Comunicação entre processo principal e seus filhos

O processo principal espera por uma mensagem

```
worker.on('message', (msg) ⇒ {
  console.log(`Message received\
    from worker process: ${msg}`);
}
```

Comunicação entre processo principal e seus filhos

Um worker espera por uma mensagem

```
cluster.worker.on('message', (msg) ⇒ {
  console.log(`Message received from master\
    process: ${msg}`);
}
```

Comunicação entre processo principal e seus filhos

Um worker envia uma mensagem ao processo principal

```
process.send('message');
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Printando

```
clustering -> node app.js
Message from worker 1: Hello master, I am the worker 1!
Message from master received by worker 1: Hello worker 1, I am your master!
Message from worker 2: Hello master, I am the worker 2!
Message from master received by worker 2: Hello worker 2, I am your master!
Message from worker 5: Hello master, I am the worker 5!
Message from worker 3: Hello master, I am the worker 3!
Message from worker 4: Hello master, I am the worker 4!
Message from master received by worker 5: Hello worker 5, I am your master!
Message from master received by worker 3: Hello worker 3, I am your master!
Message from master received by worker 4: Hello worker 4, I am your master!
Message from worker 6: Hello master, I am the worker 6!
Message from worker 7: Hello master, I am the worker 7!
Message from master received by worker 6: Hello worker 6, I am your master!
Message from master received by worker 7: Hello worker 7, I am your master!
Message from worker 8: Hello master, I am the worker 8!
Message from master received by worker 8: Hello worker 8, I am your master!
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Printando

```
clustering -> node app.js
Message from worker 1: Hello master, I am the worker 1!
Message from master received by worker 1: Hello worker 1, I am your master!
Message from worker 2: Hello master, I am the worker 2!
Message from master received by worker 2: Hello worker 2, I am your master!
Message from worker 5: Hello master, I am the worker 5!
Message from worker 3: Hello master, I am the worker 3!
Message from worker 4: Hello master, I am the worker 4!
Message from master received by worker 5: Hello worker 5, I am your master!
Message from master received by worker 3: Hello worker 3, I am your master!
Message from master received by worker 4: Hello worker 4, I am your master!
Message from worker 6: Hello master, I am the worker 6!
Message from worker 7: Hello master, I am the worker 7!
Message from master received by worker 6: Hello worker 6, I am your master!
Message from master received by worker 7: Hello worker 7, I am your master!
Message from worker 8: Hello master, I am the worker 8!
Message from master received by worker 8: Hello worker 8, I am your master!
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Final

```
if (cluster.isMaster) {
  for (let i = 0; i < numCores; i++) {
    const worker = cluster.fork();
    worker.send(`Hello worker ${worker.id}, I am your master!`);
    worker.on('message', (msg) => {
      console.log(`Message from worker ${worker.id}: ${msg}`);
    });
  }
} else {
  cluster.worker.on('message', (msg) => {
    console.log(`Message from master received by worker ${cluster.worker.id}: ${msg}`);
  });
  process.send(`Hello master, I am the worker ${cluster.worker.id}!`)
}
```

Comunicação entre processo principal e seus filhos - Printando

```
clustering -> node app.js
Message from worker 1: Hello master, I am the worker 1!
Message from master received by worker 1: Hello worker 1, I am your master!
Message from worker 2: Hello master, I am the worker 2!
Message from master received by worker 2: Hello worker 2, I am your master!
Message from worker 5: Hello master, I am the worker 5!
Message from worker 3: Hello master, I am the worker 3!
Message from worker 4: Hello master, I am the worker 4!
Message from master received by worker 5: Hello worker 5, I am your master!
Message from master received by worker 3: Hello worker 3, I am your master!
Message from master received by worker 4: Hello worker 4, I am your master!
Message from worker 6: Hello master, I am the worker 6!
Message from worker 7: Hello master, I am the worker 7!
Message from master received by worker 6: Hello worker 6, I am your master!
Message from master received by worker 7: Hello worker 7, I am your master!
Message from worker 8: Hello master, I am the worker 8!
Message from master received by worker 8: Hello worker 8, I am your master!
```

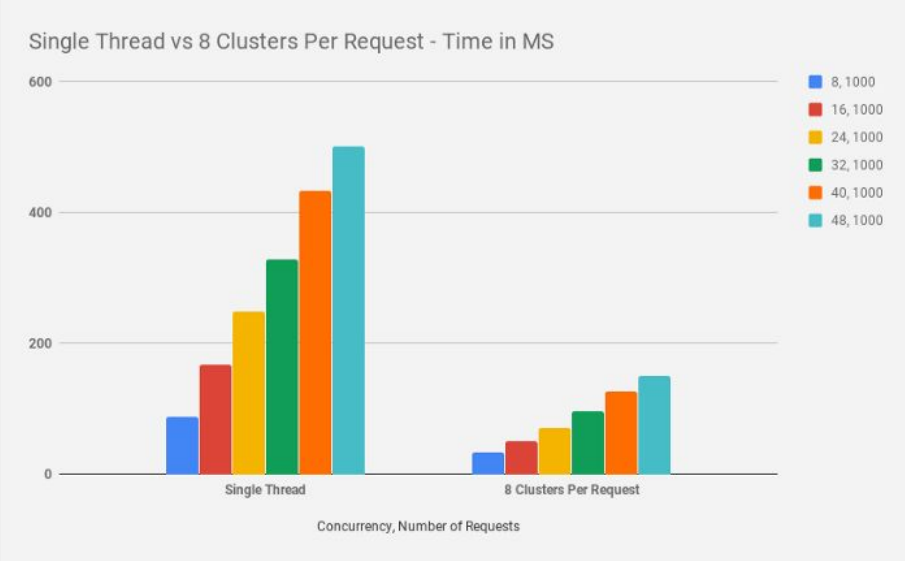
Principais desvantagens

- O gerenciamento de sessão não está disponível, as alternativas são gerenciadas por um desenvolvedor à custa da complexidade;
- IPC é um trabalho tedioso para gerenciar um aplicativo, não é a prática preferida para lidar com aplicativos complexos;
- Usar Debugger pode ser mais complicado

Benchmarking

Benchmarking - Apache benchmark

Desempenho aumentou mais de 66%.



Agora vamos analisar o módulo worker_threads

worker_threads

- Como esse módulo não gera processos filhos, um exemplo com base em requisições não é possível...

worker_threads

- Como esse módulo não gera processos filhos, um exemplo com base em requisições não é possível…
- Poderia ser mostrado um exemplo complexo, como mineração de Bitcoin ou algum algoritmo paralelizável (sort ou grafo, por exemplo), mas este tomaria muito tempo e foge do escopo desta apresentação;

worker_threads

- Como esse módulo não gera processos filhos, um exemplo com base em requisições não é possível;
- Poderia ser mostrado um exemplo complexo, como mineração de Bitcoin ou algum algoritmo paralelizável (sort ou grafo, por exemplo), mas este tomaria muito tempo e foge do escopo desta apresentação;
- Por simplicidade, vamos seguir com um exemplo simples, apenas para prova de conceito.

Cenário sem worker_threads

Cenário sem worker_threads - Código

```
console.log('Thread principal startando...');

const start = Date.now();

for (let i = 0; i < 10000; i++)
  for (let j = 0; j < 10000 * 100; j++)
    i + j;

const end = Date.now();

console.log(`Duração = ${(end - start) / 1000} segundos.`);

console.log(`Fim do script síncrono.`);
```

Cenário sem worker_threads - Código

```
console.log('Thread principal startando...');

const start = Date.now();

for (let i = 0; i < 10000; i++)
  for (let j = 0; j < 10000 * 100; j++)
    i + j;

const end = Date.now();

console.log(`Duração = ${(end - start) / 1000} segundos.`);

console.log(`Fim do script síncrono.`);
```

```
gustavo@gustavo:~/tdc$ node index.js
Thread principal startando...
Duração = 4.912 segundos.
Fim do script síncrono.
```

Cenário sem worker_threads - Código

```
console.log('Thread principal startando...');

const start = Date.now();

for (let i = 0; i < 10000; i++)
  for (let j = 0; j < 10000 * 100; j++)
    i + j;

const end = Date.now();

console.log(`Duração = ${(end - start) / 1000} segundos.`);

console.log(`Fim do script síncrono.`);
```

Note que a thread principal é bloqueada.

```
gustavo@gustavo:~/tdc$ node index.js
Thread principal startando...
Duração = 4.912 segundos.
Fim do script síncrono.
```

Cenário com worker_threads

worker_threads

Vamos dissecar esse código....

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: null });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < 1000000; i++)
    for (let j = 0; j < 10000; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

isMainThread: é *true* se o código não estiver rodando dentro de uma Worker thread.

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: null });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < 1000000; i++)
    for (let j = 0; j < 10000; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

isMainThread: é *true* se o código não estiver rodando dentro de uma *Worker* thread.

parentPort: Se a thread foi gerada como um *Worker*, ela será um *MessagePort*, permitindo a comunicação com a thread pai.

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: null });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < 1000000; i++)
    for (let j = 0; j < 10000; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

isMainThread: é *true* se o código não estiver rodando dentro de uma *Worker* thread.

parentPort: se a thread foi gerada como um *Worker*, ela será um *MessagePort*, permitindo a comunicação com a thread pai.

workerData: um valor JavaScript arbitrário que contém um clone dos dados passados para o construtor Worker.

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: null });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < 1000000; i++)
    for (let j = 0; j < 10000; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

Worker: instanciando e apontando para o próprio script (__filename).

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: 10000 });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < workerData; i++)
    for (let j = 0; j < workerData * 100; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

Worker: criando uma thread com base no próprio *script* (*__Filename)*.

workerData: variável compartilhada entre as *threads*.

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: 10000 });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < workerData; i++)
    for (let j = 0; j < workerData * 100; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

Worker: criando uma thread com base no próprio *script* (*__Filename)*.

workerData: variável compartilhada entre as *threads*.

worker.on(‘message’,...): ouvindo e esperando a thread filha responder.

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: 10000 });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < workerData; i++)
    for (let j = 0; j < workerData * 100; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

Worker: criando uma thread com base no próprio *script* (*__Filename)*.

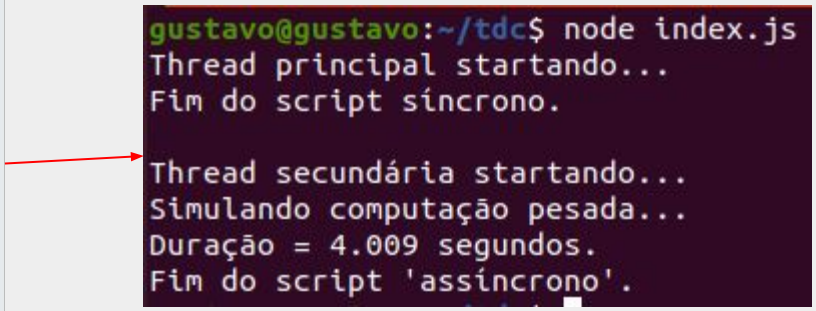
workerData: variável compartilhada entre as *threads*.

worker.on(‘message’,...): ouvindo e esperando a thread filha responder.

parentPort.postMessage: respondendo a *thread* pai.

```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: 10000 });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < workerData; i++)
    for (let j = 0; j < workerData * 100; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

```js
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: 10000 });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < workerData; i++)
    for (let j = 0; j < workerData * 100; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

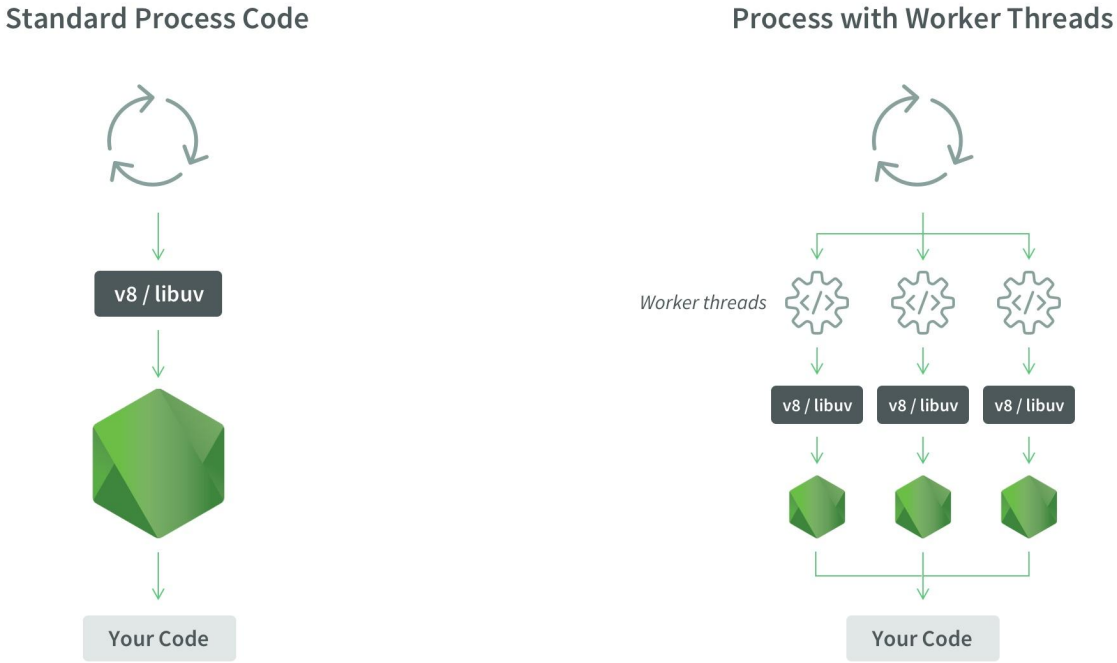


```
const {
  Worker, isMainThread, parentPort, workerData
} = require('worker_threads');

if (isMainThread) {
  console.log('Thread principal startando...');

  const worker = new Worker(__filename, { workerData: 10000 });

  worker.on('message', res => {
    console.log(`Duração = ${(res.end - res.start) / 1000} segundos.`);
  });
} else {
  console.log('Thread secundária startando...');
  console.log('Simulando computação pesada...');
  const start = Date.now();

  for (let i = 0; i < workerData; i++)
    for (let j = 0; j < workerData * 100; j++)
      i + j;

  parentPort.postMessage({ start, end: Date.now() });
}

if (isMainThread)
  console.log(`Fim do script síncrono.\n`);
if (!isMainThread)
  console.log(`Fim do script 'assíncrono'.`);
```

Note que a thread principal não é bloqueada!!

```
gustavo@gustavo:~/tdc$ node index.js
Thread principal startando...
Fim do script síncrono.

Thread secundária startando...
Simulando computação pesada...
Duração = 4.009 segundos.
Fim do script 'assíncrono'.
```

“Tentando” resumir em uma imagem



E como é em produção?


PM2

PM2 - Uma imagem vale mais que mil palavras

Cluster Mode: Node.js Load Balancing & Zero Downtime Reload

The Cluster mode is a special mode when starting a Node.js application, it starts multiple processes and load-balance HTTP/TCP/UDP queries between them. This increase overall performance (by a factor of x10 on 16 cores machines) and reliability (faster socket re-balancing in case of unhandled errors).

Starting a Node.js application in cluster mode that will leverage all CPUs available:

```
$ pm2 start api.js -i <processes>
```

`<processes>` can be `'max'`, `-1` (all cpu minus 1) or a specified number of instances to start.

https://www.npmjs.com/package/pm2

PM2 - Uma imagem vale mais que mil palavras

Cluster Mode: Node.js Load Balancing & Zero Downtime Reload

The Cluster mode is a special mode when starting a Node.js application, it starts multiple processes and load-balance HTTP/TCP/UDP queries between them. This increase overall performance (by a factor of x10 on 16 cores machines) and reliability (faster socket re-balancing in case of unhandled errors).

Starting a Node.js application in cluster mode that will leverage all CPUs available:

```
$ pm2 start api.js -i <processes>
```

`<processes>` can be `'max'`, `-1` (all cpu minus 1) or a specified number of instances to start.

https://www.npmjs.com/package/pm2

PM2 - Uma imagem vale mais que mil palavras

Cluster Mode: Node.js Load Balancing & Zero Downtime Reload

The Cluster mode is a special mode when starting a Node.js application, it starts multiple processes and load-balance HTTP/TCP/UDP queries between them. This increase overall performance (by a factor of x10 on 16 cores machines) and reliability (faster socket re-balancing in case of unhandled errors).

Starting a Node.js application in cluster mode that will leverage all CPUs available:

```
$ pm2 start api.js -i <processes>
```

`<processes>` can be `'max'`, `-1` (all cpu minus 1) or a specified number of instances to start.

https://www.npmjs.com/package/pm2

PM2 - Uma imagem vale mais que mil palavras

Cluster Mode: Node.js Load Balancing & Zero Downtime Reload

The Cluster mode is a special mode when starting a Node.js application, it starts multiple processes and load-balance HTTP/TCP/UDP queries between them. This increase overall performance (by a factor of x10 on 16 cores machines) and reliability (faster socket re-balancing in case of unhandled errors).

Starting a Node.js application in cluster mode that will leverage all CPUs available:

```
$ pm2 start api.js -i <processes>
```

`<processes>` can be `'max'`, `-1` (all cpu minus 1) or a specified number of instances to start.

https://www.npmjs.com/package/pm2

Fim!

Obrigado!

Gustavo Oliveira
Engenheiro de Software na Gympass



gustavooliveiraf

Dúvidas?

Referências

- https://medium.com/better-programming/is-node-js-really-single-threaded-7ea59bcc8d64
- https://imasters.com.br/front-end/usando-worker-threads-no-node-js
- https://blog.logrocket.com/a-complete-guide-to-threads-in-node-js-4fa3898fe74f/
- https://www.infoq.com/br/articles/nodejs-utilizando-modulo-de-cluster/
- https://nodesource.com/blog/worker-threads-nodejs/