05

Escalabilidade e Clusters

Escalabilidade

O PayFast já está pronto para ir para a produção. E nesse momento é bem provável que até já tenha sido deployado. Agora é a hora de se preocupar com novas questões. Um delas é o que fazer quando o serviço atingir seu limite de processamento devido a um elevado número de clientes consumindo.

Esse é um problema com o qual todos queremos nos preocupar, pois significa que o sistema está tendo bastante acesso e está ganhando escala. A principal atitude que devemos ter nesse momento é verificar e implementar as estratégias disponível para escalar bem a aplicação de acordo com as limitações que temos de software e hardware.

Cluster

Com relação ao software, sabemos que não é possível utilizar mais de uma *thread*, no Node. O que é bastante criticado por muitas pessoas, porém este não é necessariamente um problema tão grave para ele, visto que ele consegue escalonar seus processos com uma velocidade realmente muito boa, mesmo utilizando somente uma thread.

Além disso, o fato de não poder ter mais de uma thread não é impeditivo para que se crie um **cluster** para uma aplicação Node. Na verdade, o Node até já vem com uma lib nativa em seu *core* especialista no assunto: o módulo **cluster**.

Ao utilizarmos esse módulo, ele basicamente instancia novos processos de uma aplicação, trabalhando de forma distribuída e, quando trabalhamos com uma aplicação web, esse módulo se encarrega de compartilhar a mesma porta da rede entre os clusters ativos. O número de processos a serem criados é determinado pelo programador, e é claro que a boa prática é instanciar um total de processos relativo à quantidade de núcleos do processador do servidor, ou também uma quantidade relativa a núcleos X processadores.

Por exemplo, se a máquina possui apenas um processador de 4 núcleos, então é possível instanciar 4 processos, criando assim uma rede com 4 nós. Mas caso tenha 4 processadores de 4 núcleos cada, então é possível criar 16 processos, tendo assim uma rede com 16 nós ativos.

Para garantir que os *clusters* trabalhem de forma organizada e distribuida é preciso que haja um *cluster master*. Ele é o processo pai, a partir do qual todos os outros são criados e é responsável por balancear a carga entre os filhos, que são conhecidos como *cluster slaves*.

Uma grande vantagem de implementar esse tipo de arquitetura no Node é que toda a parte de criação dos *clusters* e distribuição dos processos fica muito bem abstraida, tornando a implementação bastante simples. Outra é que a execução dos *clusters* é independente, o que significa que se um deles cair, os outros continuam a executar normalmente. Porém a capacidade de tentar trazer de volta à ativa esse *cluster* perdido, essa sim precisa ser feita manualmente.

Vamos criar na raíz do projeto PayFast um arquivo chamado `cluster.js` com o intuito de que este passe a ser agora o arquivo que inicia a aplicação e que faça isso implementando *clusters*

O primeiro passo é carregar a lib **cluster**:

```
var cluster = require ('cluster');
```

A partir dessa variável `cluster` é possível agora criar os novos *clusters*. Porém como sabemos quantos *clusters* podemos criar? Essa informação deve vir do sistema operacional, que é quem conhece a arquitetura física da máquina onde ele está rodando. Para obter informações do SO, existe uma lib com exatamente esse nome, que também faz parte do core do Node. Ela dever ser carregada também.

```
var cluster = require ('cluster');
var os = require('os');
```

Agora podemos verificar quantos cpus existem e criar um novo *cluster slave* para cada:

```
var cluster = require ('cluster');
var os = require('os');

const CPUS = os.cpus();

CPUS.forEach( function(){
  cluster.fork()
});
```

Porém se fizermos dessa forma, teremos um problema, pois cada novo *cluster* criado também executará esse código no momento de sua criação. Ou seja todos executarão o código `cluster.fork()` uma vez para cada cpu da máquina. Isso significa que serão criados infinitos *clusters* até que a máquina não suporte e trave o processamento.

O que nós queremos é eleger somente um *cluster master* para que seja o único capaz de criar outros *clusters* e que cada um desses novos *clusters* ao serem criados, executem o arquivo `index.js` que é o arquivo que sobe o **express**, ao invés de criarem também seus próprios filhos de maneira infinita:

```
var cluster = require ('cluster');
var os = require ('os');

const CPUS = os.cpus();

// A primeira coisa é verificar se é o master
// pois ele é o único que pode invocar o fork()
if (cluster.isMaster) {

  CPUS.forEach(function(){
    cluster.fork()
  });

// Se o cluster não for o master, ele deve somente executar o index.js
// subindo o objeto do express e ficando no ar para receber requisições.
} else {
  require('./index.js');
}
```

Esse é o básico para criar *clusters* com Node. É realmente bastante simples e muito útil! A partir daí podemos ainda melhorar a gestão da rede, criando *listeners*, por exemplo para informar que um novo *cluster* está ativo ou inativo e ainda tentar recolocar no ar, algum que tenha caído inesperadamente.

```
var cluster = require ('cluster');
var os = require('os');

const CPUS = os.cpus();

if (cluster.isMaster) {

  CPUS.forEach(function(){
    cluster.fork()
  });

  cluster.on("listening", worker => {
    console.log("cluster %d conectado", worker.process.pid);
  });

  cluster.on("disconnect", worker => {
    console.log("cluster %d desconectado", worker.process.pid);
  });

  cluster.on("exit", function(worker)  {
    console.log("cluster %d perdido", worker.process.pid);
    cluster.fork();
  });

} else {
  require('./index.js');
}
```