Node.js – O que é, como funciona e quais as vantagens

05/09/2018 / por lenon

O Node.js pode ser definido como um **ambiente de execução Javascript *server-side***.

Isso significa que com o Node.js é possível criar aplicações Javascript para rodar como uma aplicação *standalone* em uma máquina, não dependendo de um browser para a execução, como estamos acostumados.

Apesar de recente, o Node.js já é utilizado por grandes empresas no mercado de tecnologia, como Netflix, Uber e LinkedIn.

O principal motivo de sua adoção é a sua **alta capacidade de escala**. Além disso, sua arquitetura, flexibilidade e baixo custo, o tornam uma boa escolha para implementação de Microsserviços [https://www.opus-software.com.br/micro-servicos-arquietura-monolitica/] e componentes da arquitetura *Serverless* [https://www.opus-software.com.br/serverless-applications/]. Inclusive, os principais fornecedores de produtos e serviços

Neste artigo, vamos trazer a história, características e vantagens que tornam o Node.js uma tecnologia única.

O surgimento do Node.js

Apesar do Javascript ter mais de 20 anos, o seu uso *server-side* é bem recente.

A linguagem Javascript foi criada em 1995, e se tornou a linguagem padrão dos *browsers* e consequentemente da *Web* para o desenvolvimento *client-side*.

Desde então, houveram diversas tentativas de implementar sua execução *server-side*. Todas elas fracassaram, devido à sua performance ser extremamente baixa comparado com as linguagens existentes no mercado, como o PHP ou Java.

Porém, com a rápida evolução da *Web* nos últimos anos, a linguagem Javascript e seus motores de execução passaram por diversas melhorias, tornando viável sua execução com outros propósitos além da manipulação de páginas HTML.

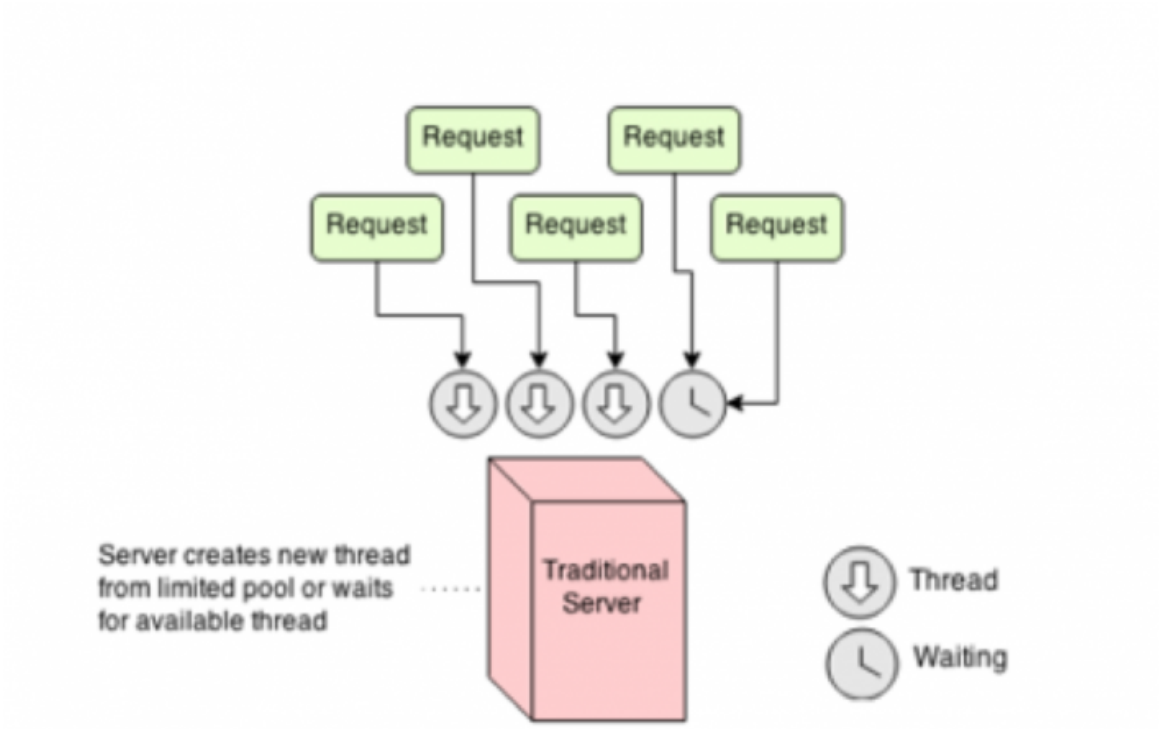
Com essa nova fase no uso do Javascript, aplicações *server-side* passaram a ser implementadas, e em 2009 foi criado o primeiro ambiente de execução Javascript com este propósito: **O Node.js**.

Características

A principal característica que diferencia o Node.JS de outras tecnologias, como PHP, Java, C#, é o fato de **sua execução ser *single-thread***. Ou seja, apenas uma thread é responsável por executar o código Javascript da aplicação, enquanto que nas outras linguagens a execução é *multi-thread*.

requisição recebida é criada uma nova *thread* para trata-la. A cada requisição, serão demandados recursos computacionais (memória RAM, por exemplo) para a criação dessa nova *thread*. Uma vez que esses recursos são limitados, as *threads* não serão criadas infinitamente, e quando esse limite for atingido, as novas requisições terão que esperar a liberação desses recursos alocados para serem tratadas.

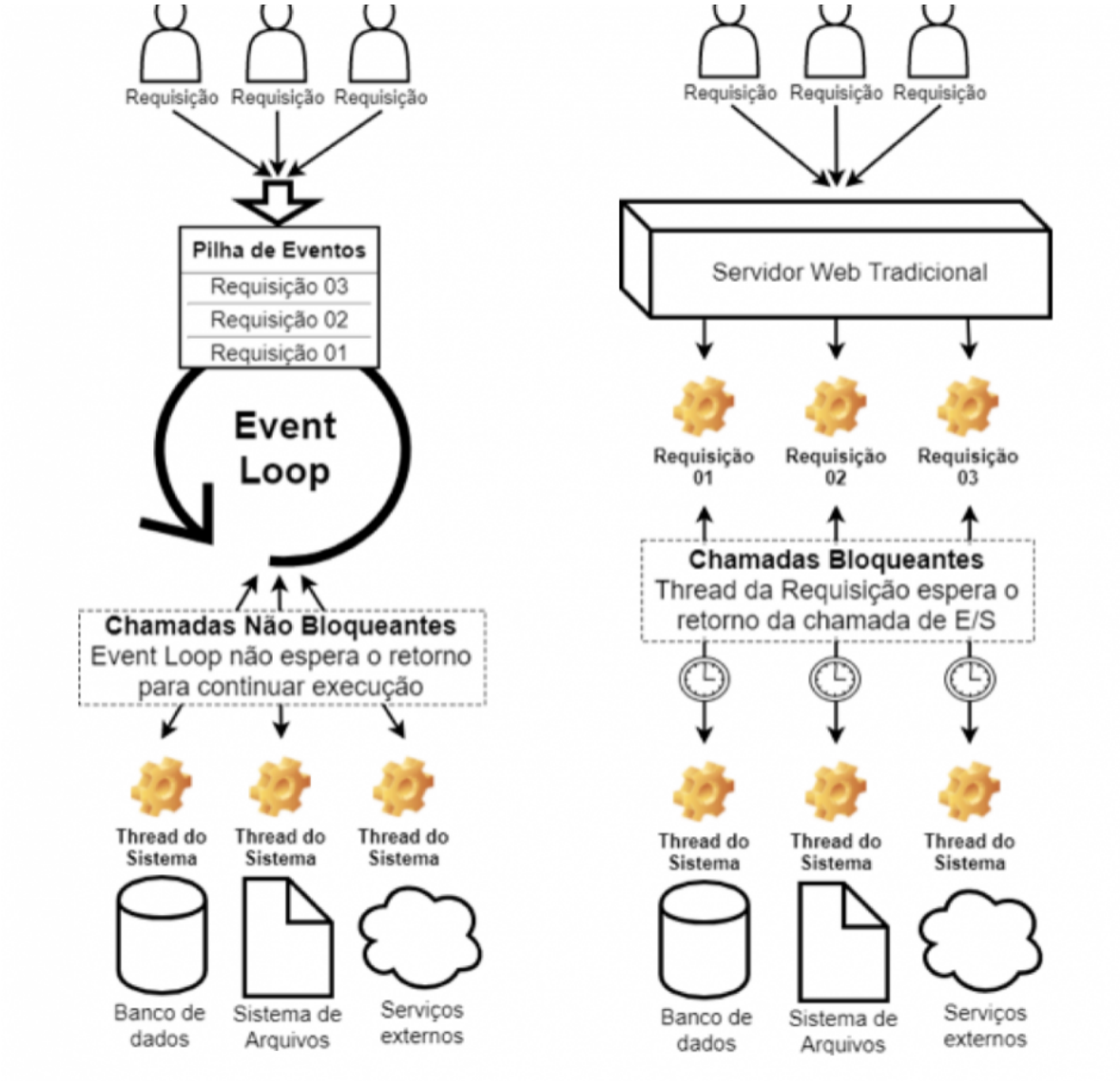
A figura abaixo representa esse cenário em um servidor tradicional:



No modelo Node.js, apenas uma *thread* é responsável por tratar as requisições. Essa thread é chamada de ***Event Loop***, e leva esse nome pois cada requisição é tratada como um evento. O *Event Loop* fica em execução esperando novos eventos para tratar, e para cada requisição, um novo evento é criado.

mesmo efeito através de chamadas de E/S (entrada e saída) **não-bloqueantes**. Isso significa que as operações de entrada e saída (ex: acesso a banco de dados e leitura de arquivos do sistema) são assíncronas e não bloqueiam a *thread*. Diferentemente dos servidores tradicionais, a *thread* não fica esperando que essas operações sejam concluídas para continuar sua execução.

A figura abaixo representa a diferença de funcionamento de um servidor web tradicional e um Node.JS:



No servidor Node.js, o *Event Loop* é a única *thread* que trata as requisições, enquanto que no modelo tradicional uma nova *thread* é criada para cada requisição. Enquanto o *Event Loop* delega uma operação de E/S para uma *thread* do sistema de forma assíncrona e continua tratando as outras requisições que aparecerem em sua pilha de eventos, as *threads* do modelo tradicional esperam a conclusão das operações de E/S, consumindo recursos computacionais durante todo esse período de espera.

Apesar do Node.js ser *single-threaded*, sua arquitetura possibilita um número maior de requisições concorrentes sejam tratadas em comparação

Vantagens de uso do Node.js

Flexibilidade

O NPM (Node Package Manager) é o gerenciador de pacotes do Node.js e também é o maior repositório de softwares do mundo. Isso faz do Node.js uma plataforma com potencial para ser utilizada em qualquer situação. O pacote mais conhecido se chama Express.js e é um framework completo para desenvolvimento de aplicações Web.

Leveza

Criar um ambiente Node.js e subir uma aplicação é uma tarefa que não exige muitos recursos computacionais em comparação com outras tecnologias mais tradicionais. Se utilizado em conjunto com ferramentas como o Docker, o ganho na velocidade de *deploy* e replicação de máquinas pode ser muito significativo e em ambientes escaláveis isso significa menos custo e mais eficiência.

Tanto sua leveza quanto flexibilidade fazem do Node.JS uma tecnologia indicada para a implementação de serviços e componentes de arquiteturas como a de **microsserviços e serverless**. Além disso, conta com suporte das principais empresas de produtos e serviços Cloud do mercado, como a AWS, Google Cloud e Microsoft Azure que oferecem na maioria de seus produtos suporte nativo ao Node.JS.

Produtividade da equipe

Maior repositório do mundo: O NPM fornece pacotes de código reusáveis e provavelmente aquela integração que você precisa fazer com outro sistema ou banco de dados já está implementado e disponível gratuitamente para instalar via NPM.

partida importante para iniciar o uso do Node.js. Além disso, esse fator pode representar ganhos de reutilização de código e criação de equipes multidisciplinares, com melhor aproveitamento de recursos.

Ambiente de inovação: Possibilidade de deploys e iterações mais rápidas, e resolução de problemas *On the Fly*. Isso também permite a criação de soluções próprias e inovadoras, como fez o Uber [https://foundation.nodejs.org/wp-content/uploads/sites/50/2017/09/Nodejs-at-Uber.pdf] criando produtos em Node.js para resolver alguns de seus problemas.

Casos de uso mais comuns

Aplicações em Tempo Real

Um exemplo comum é uma aplicação de conversa (chat). Tal aplicação exige muito pouco processamento e basicamente consiste em transferir as mensagens de um lado para outro.

Ambientes Escaláveis

O Node.js é bastante indicado para ambientes escaláveis (com grande número de conexões concorrentes), já que tem potencial para suportar um número maior de conexões simultâneas do que servidores tradicionais.

Camada de Entrada do Servidor

O Node.js faz pouco processamento de dados e apenas passa a requisição para frente, se comunicando com serviços de *backend*.

Mocks e Protótipos

externo, por exemplo.

API com NoSQL por trás

As base de dados NoSQL são baseadas em JSON (*JavaScript Object Notation*), portanto, sua comunicação com Node.js é bastante intuitiva. Com isso, não é necessário converter modelos de dados, por exemplo, pois os mesmos objetos JavaScript armazenados na base de dados podem ser enviados para o front-end sem a necessidade de nenhum tipo de tratamento ou conversão.