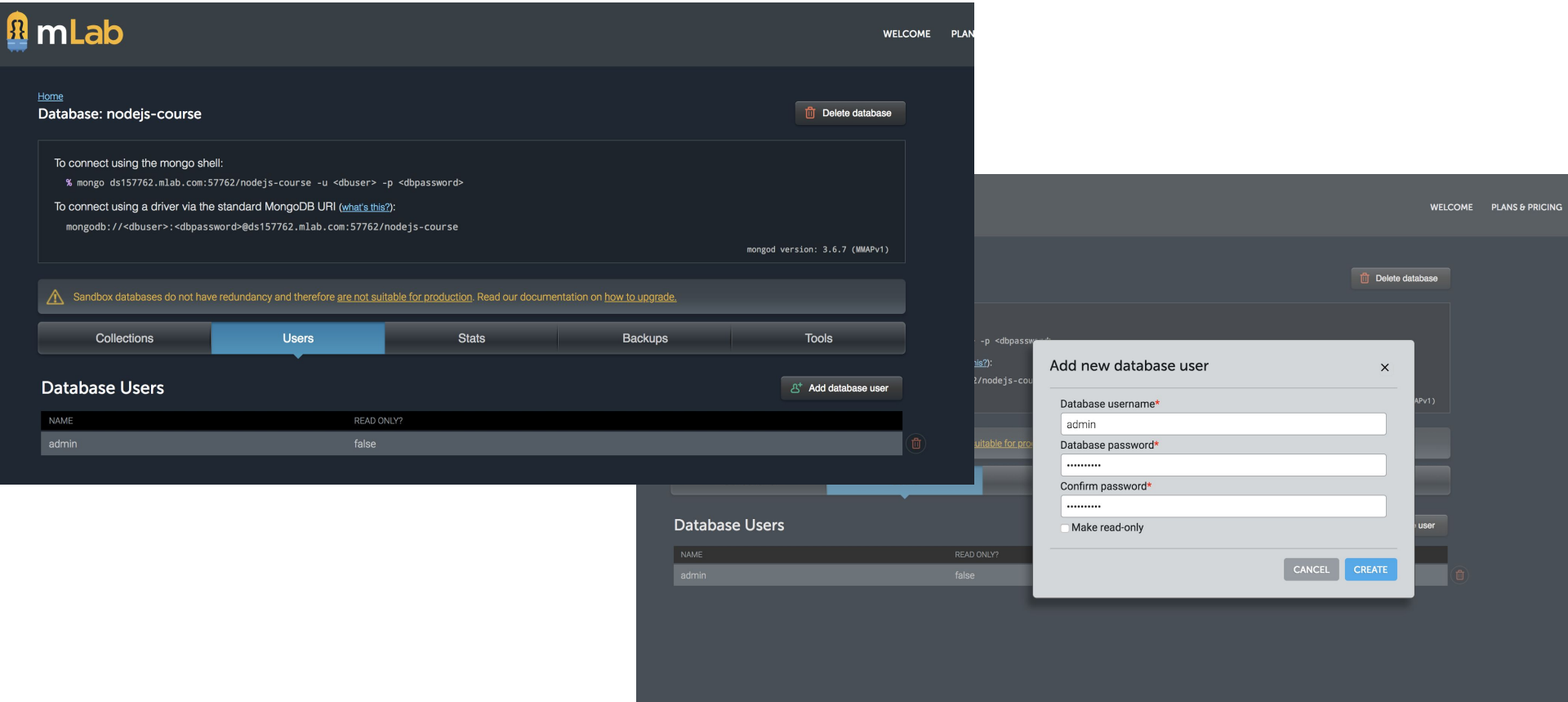
Criar uma conta nos seguintes websites

- TravisCI (Integração contínua)
  - https://travis-ci.org
- MLab (Banco de Dados)
  - https://mlab.com/
- Heroku (Plataforma de aplicações em nuvem)
  - http://heroku.com/
- Instalar "nodemon" globalmente
  - **npm install -g nodemon**

Adicione um usuário para o database no site mlab



APIs testáveis e escaláveis de alta performance com nodejs

Renan Pupin
André Andreotti

Sobre

- Renan
  - Bacharel em Ciência da Computação pela FCT-UNESP, 2016
  - Aluno especial do Programa de Mestrado em Ciência da Computação
  - CTO e Fundador do FalaFreud, 2016
- André
  - Bacharel em Ciência da Computação pela FCT-UNESP, 2016
  - Mestrando em Ciência da Computação pelo IBILCE-UNESP, 2017
  - Pesquisador no projeto Radiar na Petrobrás, 2017
  - Desenvolvedor no FalaFreud, 2018

Sumário

- Javascript
- Nodejs
- APIs REST
- MongoDB e No-SQL
- Integração contínua e Cloud Computing
- TDD e Escalabilidade

Sumário

- Hands On
  - Hello World node.js
  - Desenvolvendo uma API orientada a testes
  - Executando sua API para a nuvem e disponibilizando para a internet
  - Integrar a um CI e rodar testes automatizados automaticamente

Javascript

O que é?

- Linguagem de programação
- Brendan Eich, 1995 - Mozilla Corporation
- LiveScript
- Rebatizada para JavaScript
- Orientada a objetos
- Navegadores, Photoshop, GNOME, Node.js

Características

- Várias funcionalidades avançadas
- Pequeno esforço
- APIs disponíveis em navegadores web
- APIs de terceiros (twitter, facebook...)
- Frameworks e bibliotecas

Sintaxe

- Case-Sensitive
- Unicode (107 mil caracteres)
- Instruções separadas por ponto e vírgula (;)
- Palavras reservadas (if, else, case, break...)
- (a.if, a[‘if’], a = {if : ‘test’}) são permitidas pois não são identificadores

Tipos, estrutura e JSON

- var (declara uma variável que é acessível independente do escopo de bloco)
- let (declara uma variável local de escopo de bloco)
- const (declara uma constante apenas de leitura)
- Tipos de dados (Boolean, null, undefined, Number, String, Symbol (ES6), Object, Array)
- Dinamicamente tipada

```
1  var a = 7;
2  a = "Isso é uma String.";
```

Tipos, estrutura, JSON

- JSON (JavaScript Object Notation)
- Sintaticamente idêntico ao código para criar objetos JavaScript
- Utilizado para troca de dados do servidor com a página web

```
{
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
}
```

ES6 - ECMAScript 6

- Ser uma linguagem melhor para construir aplicações complexas
- Resolver problemas antigos do JavaScript
- Facilidade no desenvolvimento de *libraries*

ES6 - ECMAScript 6

- Inclusão de “let” e “const” na declaração de variáveis
- Arrow Functions

(a)

```
var sum = function(x, y) {
  return x + y;
};
```

(b)

```
const sum = (x, y) => {
  return x + y
}
```

(c)

```
const sum = (x, y) => x + y
```

ES6 - ECMAScript 6

- Utilizando o “this”

```
1  function Widget() {
2      var button = document.getElementById('button');
3      button.addEventListener('click', function() {
4          this.doSomething();
5      });
6  }
```

- O 'this' não aponta para Widget como esperado e provocará um erro

ES6 - ECMAScript 6

- Utilizando o “this” com arrow function

```
1  function Widget() {
2      const button = document.getElementById('button')
3      button.addEventListener('click', () => {
4          this.doSomething()
5      })
6  }
```

- O 'this' aponta para Widget e não provocará nenhum erro

ES6 - ECMAScript 6

- Default parameters

```
1  const multiply = (x, y = 1) => {
2    return x * y
3  }
4
5  multiply(3, 2) // 6
6  multiply(3) // 3
```

Closures

- add5 e add10 são closures
- makeAdder é uma função fábrica

```
1  function makeAdder(x) {
2    return function(y) {
3      return x + y;
4    };
5  }
6
7  var add5 = makeAdder(5);
8  var add10 = makeAdder(10);
9
10 print(add5(2));  // 7
11 print(add10(2)); // 12
```

Módulos

- Organizar o sistema
- Aumentar o reuso
- Diminuir a complexidade

Módulos

- Exportar módulos

```
1  //lib.js
2  exports.sum = function (x, y) {
3      return x + y;
4  }
```

- Importar módulos

```
1  const math = require('./lib.js')
2
3  math.sum(1, 2) //3
```

Callbacks

- Função passada como argumento para outra função
- Invocada dentro da função externa para completar algum tipo de rotina

```
1  function greeting(name) {
2    alert('Hello ' + name);
3  }
4
5  function processUserInput(callback) {
6    var name = prompt('Please enter your name.');
7    callback(name);
8  }
9
10 processUserInput(greeting);
```

Promises

- Um objeto que representa a eventual conclusão ou falha de uma operação assíncrona

```
1  let p = new Promise (function (resolve, reject) {
2      // resolva ou rejeite
3  });
```

Promises

Com callback

```
doSomething(function(result) {
  doSomethingElse(result, function(newResult) {
    doThirdThing(newResult, function(finalResult) {
      console.log('Got the final result: ' + finalResult);
    }, failureCallback);
  }, failureCallback);
}, failureCallback);
```

Promises

Com promise

```
1  doSomething().then(function(result) {
2    return doSomethingElse(result);
3  })
4  .then(function(newResult) {
5    return doThirdThing(newResult);
6  })
7  .then(function(finalResult) {
8    console.log('Got the final result: ' + finalResult);
9  })
10 .catch(failureCallback);
```

Nodejs

Node.js® is a JavaScript runtime platform built on Chrome's V8 JavaScript engine.

História

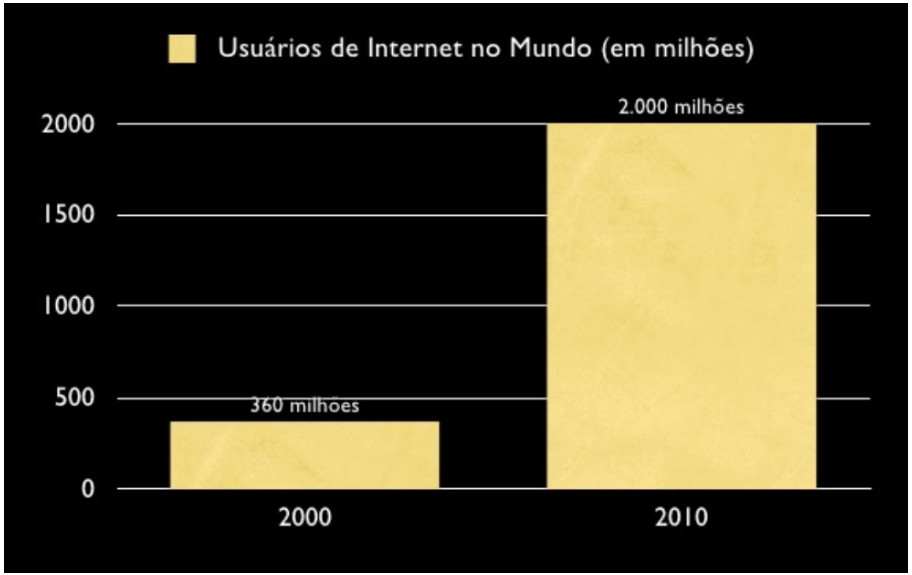
- Criado por **Ryan Dahl** em 2009
- **Interpretador** de código **JavaScript**
- Open Source
    - **Atualmente** seu desenvolvimento é **mantido** pela **Node.js Foundation** em parceria com a **Linux Foundation**.

Sobre o nodejs

- Desenvolvimento **Javascript no lado do servidor**
- Plataforma de desenvolvimento de **aplicações** altamente **escaláveis**, com códigos **capazes** de **manipular** dezenas de **milhares** de **conexões simultâneas**, numa única máquina física.
- **Multiplataforma**
- **Independente** de **navegador**

Motivação

- **Anos 2000**: **360 milhões** de pessoas com acesso a internet.
- Em **2010**: **2 bilhões**



PHP	1995
Java EE	1998
ASP.Net	2002
Ruby on Rails	2005
Django	2005

Motivação

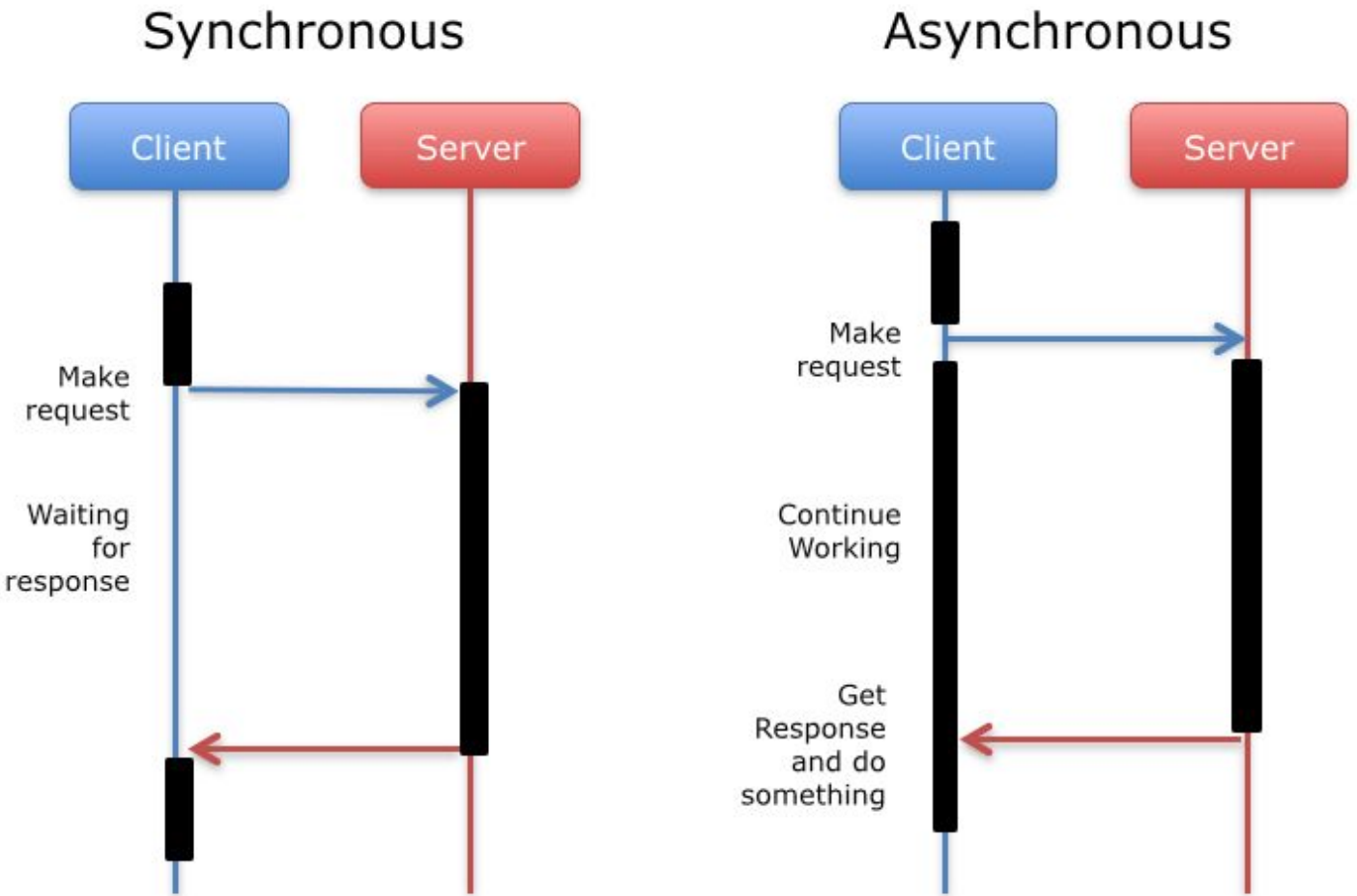
- Código costumava ser escrito assim

```
findUserByName(name){
    var busca = "SELECIONE USUARIO ONDE NOME = 'NOME'";
    var usuario = Usuario.encontre(busca);

    retorne usuario;
}
```

- O que o software está fazendo enquanto a busca executa?
- Na maioria dos casos está **travado esperando** a **resposta** do banco

Motivação

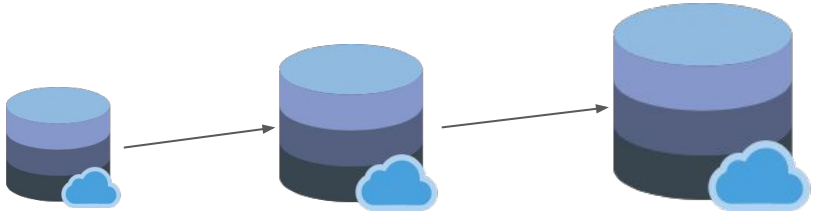
- **Servidor web** com requisições de **I/O bloqueantes**
  - Leitura de disco, query no banco de dados, envio de email, requisição para API de terceiros
- **Uma Thread** para **cada conexão**
  - Se **uma Thread usa 2mb**
  - **2mb x 1000** conexões simultâneas = **2GB**
- Como manter conectados **100 mil usuários simultâneos**?
  - 2mb x 100.000 conexões simultâneas = **200GB????**



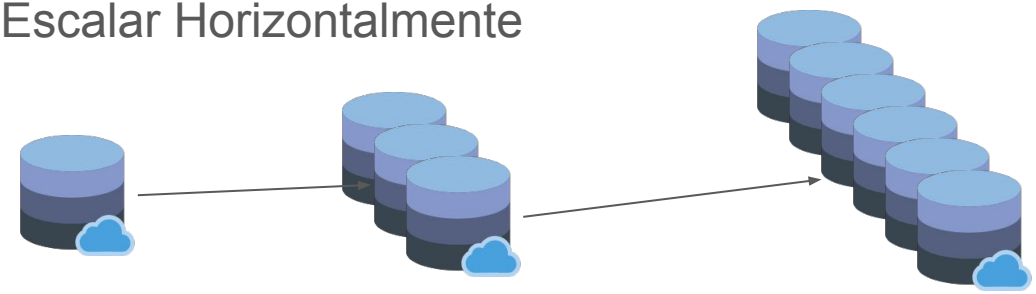
https://medium.com/@santhoshhari/efficient-web-scraping-with-pythons-asynchronous-programming-6b9e730f1ff7

Possíveis soluções

- Escalar Verticalmente



- Escalar Horizontalmente



- Desenvolver uma linguagem de programação mais rápida

Solução: Novos Paradigmas

- **Programação orientada a evento**
- **Single Thread**
  - **Múltiplas Threads não funcionam bem** com eventos de **I/O**
  - **Threads escalam** em utilização de **CPU** (Processamento Paralelo) mas comprometem a memória
- **Requisições I/O**
  - **Assíncronas**
  - **Não bloqueantes**

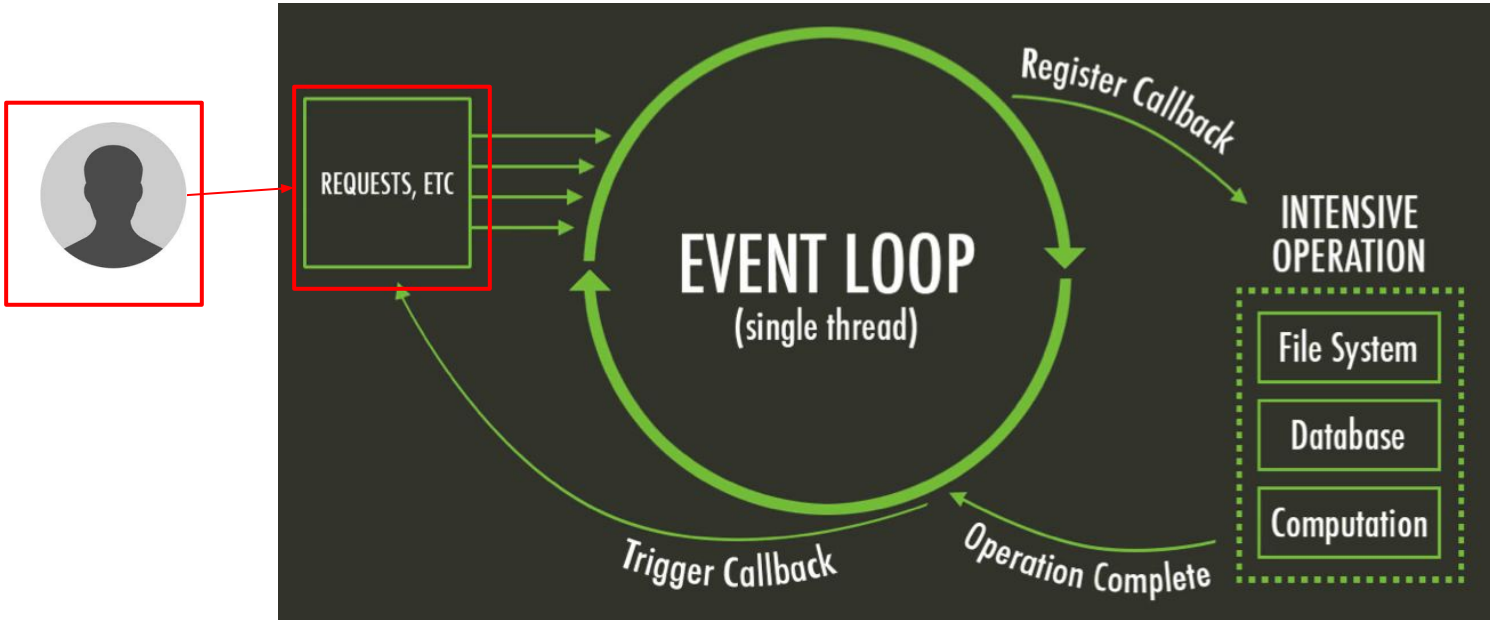
Como funciona?

- No Nodejs as **requisições** são **processadas uma por vez**
- O Nodejs **opera** em **single-thread** puramente **baseado** em **eventos**
- Usa o **modelo** de **I/O** com infraestrutura **não bloqueante**, **eficiente** e altamente **escalável**

O fluxo

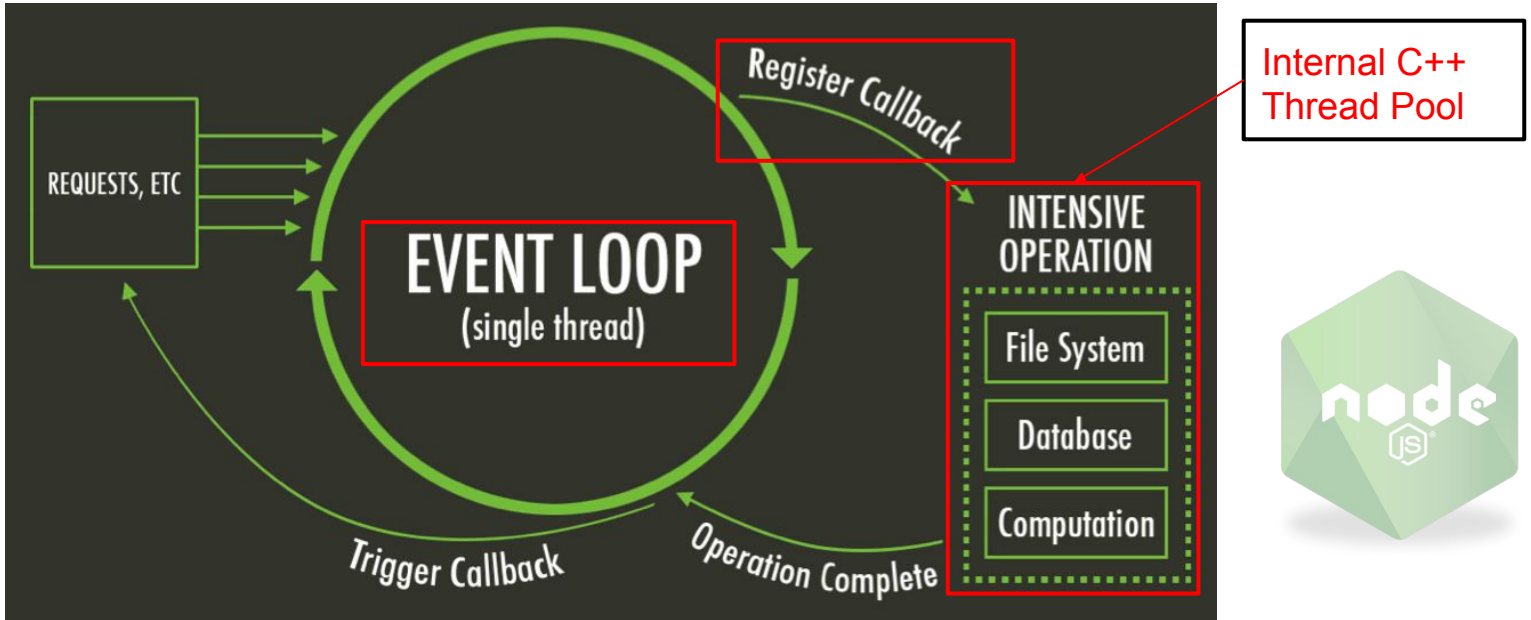
- O **usuário envia** uma **requisição** para o **servidor** nodejs

https://stackoverflow.com/questions/21596172/what-function-gets-put-into-eventloop-in-nodejs-and-js



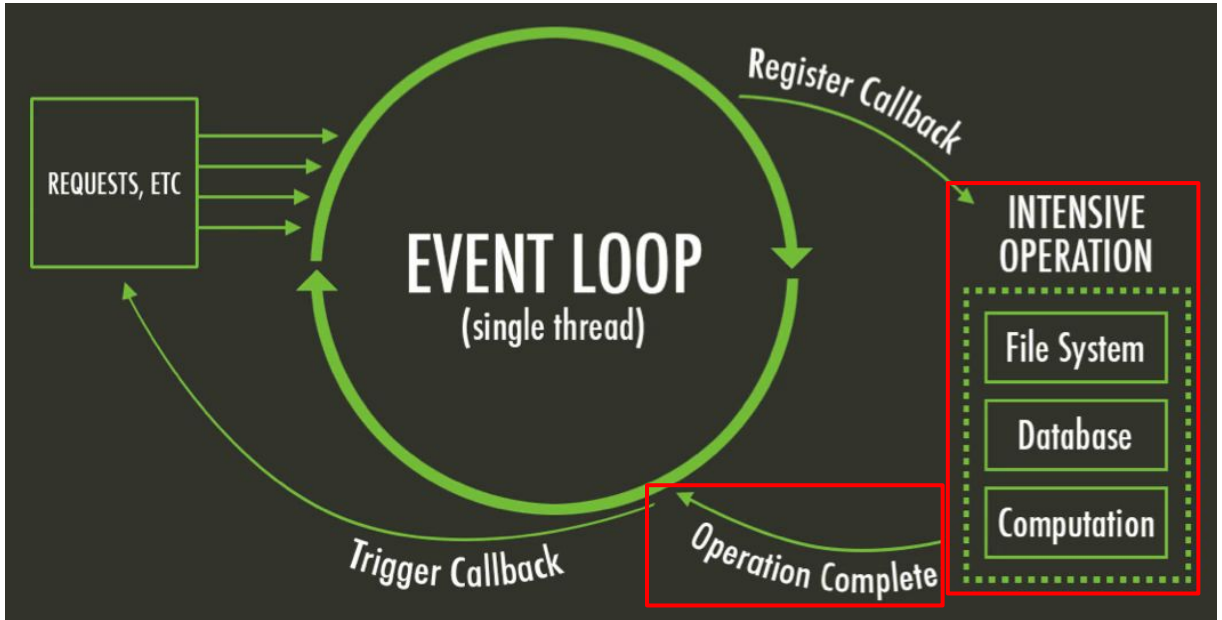
O fluxo

- O **loop de eventos** da thread principal **registra** um **callback**, encaminha o trabalho para um **pool de threads** e imediatamente **retorna** para **escutar** a **próxima conexão**.



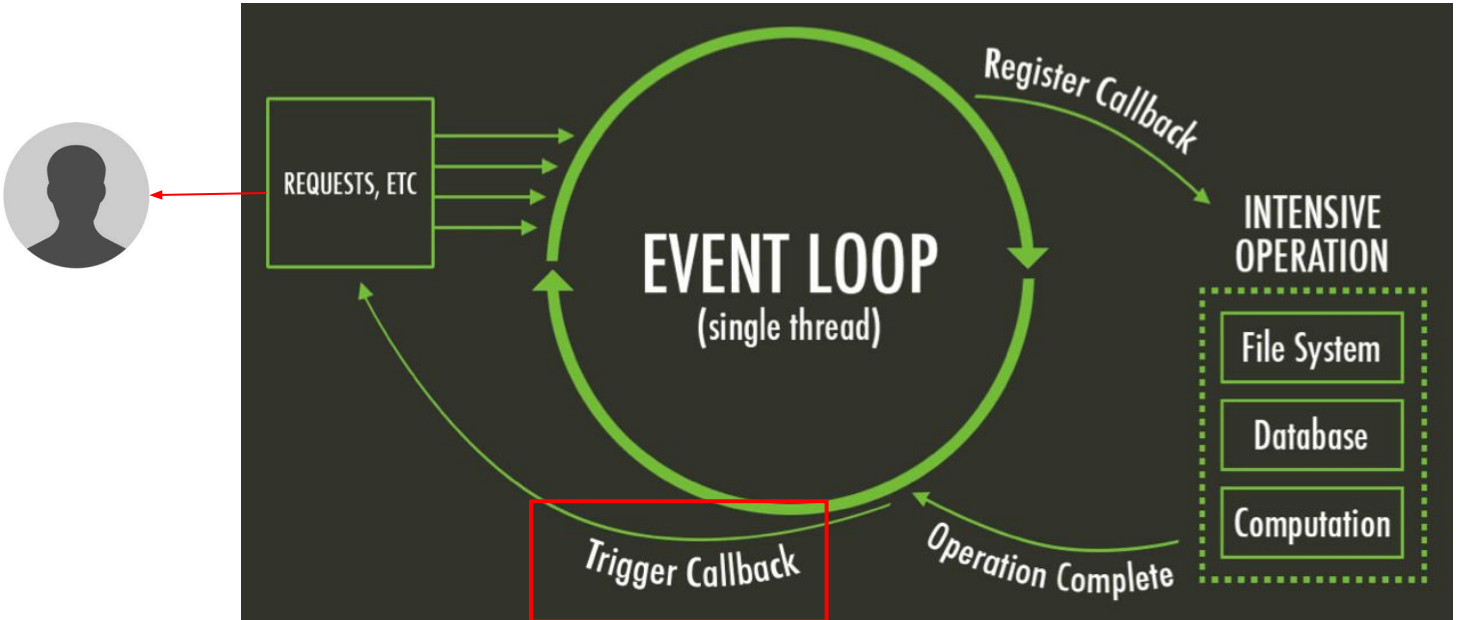
O fluxo

- As **threads do pool** executam as **tarefas** de **longa duração** de forma **assíncrona** e **paralelizada**. Ao **finalizar** uma **tarefa**, a **resposta** é **enviada** para a **thread principal** via **callback**.



O fluxo

- O **loop de eventos retorna** o **resultado** para o **usuário**.



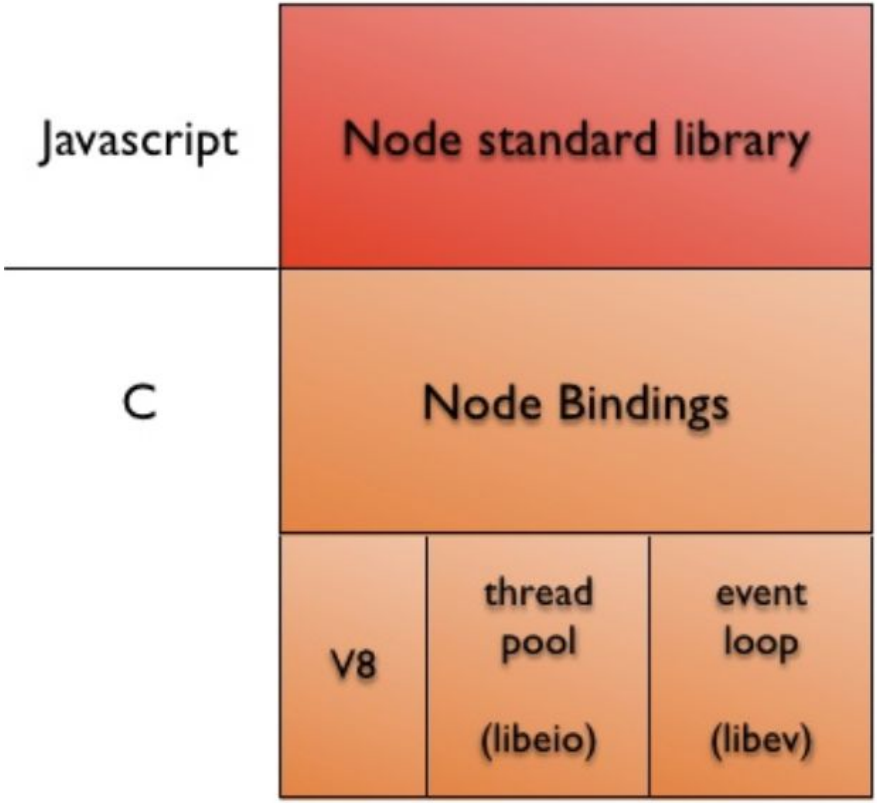
Sobre o nodejs

- **Porque não fazíamos isso antes?**
  - **I/O assíncrono não era suportado em todos os SOs**
- **Porque Javascript?**
  - **JS** já foi **estruturado** pensando em ser **assíncrono**

Arquitetura

- Escrito em C/C++ e javascript

- libuv - Cross-platform asynchronous I/O
  - http://libuv.org/



Sobre o nodejs

- Nesse exemplo de "Hello World"
  - Várias conexões podem ser gerenciadas simultaneamente
  - Para cada conexão um callback é retornado
  - Porém se não tiver trabalho a ser feito,

    o nodejs vai "dormir"
- https://nodejs.org/

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

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Quem usa?

Vantagens

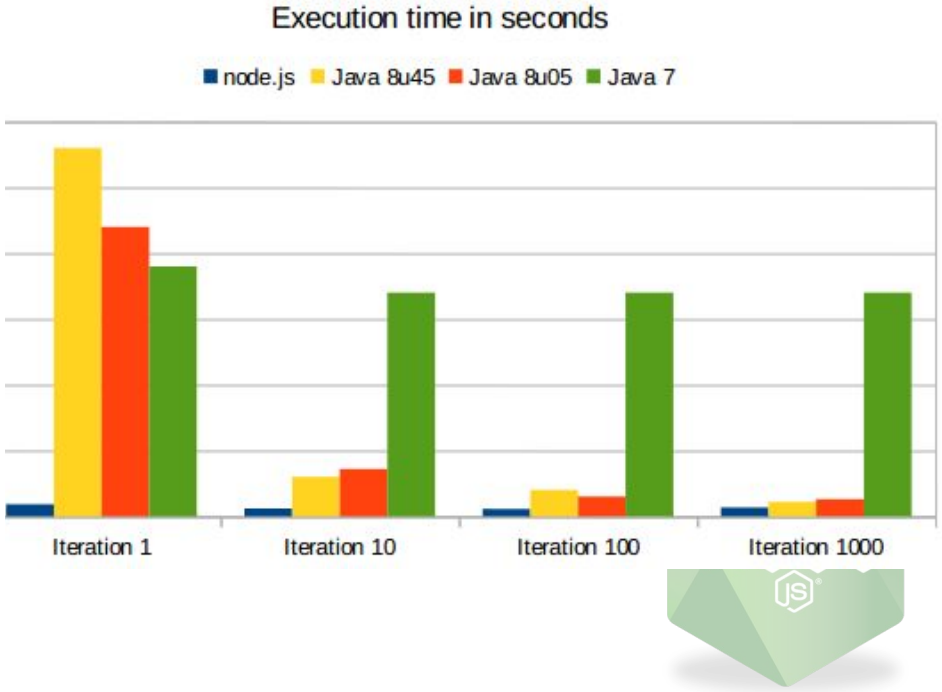
- Node é totalmente **modular** (na verdade ele é um conjunto de módulos)
- **Simples** (basta abrir o terminal e digitar o comando "node")
- É **otimizado** para aplicações **real time**
- Permite **gerenciar** um **alto nível** de **concorrência** sem **sobrecarregar** o **CPU**
- **Multiprotocolo** (HTTP, DNS, TLS, etc)
- É **possível** criar um **cluster** e trabalhar em **multi-threading**
- **Funciona bem** com **banco de dados não-relacionais**

Quando não usar node?

- Quando for preciso fazer **codificação** de **vídeo** e **processamento intenso**
  - O grande uso de **CPU anula** os **benefícios** do **I/O não bloqueante**
  - Assim todas as **outras conexões** ficarão **bloqueadas** enquanto a **thread está ocupada** com o **processamento**
- Quando for preciso **utilizar** muitos **módulos específicos**
  - Nodejs é recente
- Se você confunde **java** com **#javascript** xD

Comparativo  vs 

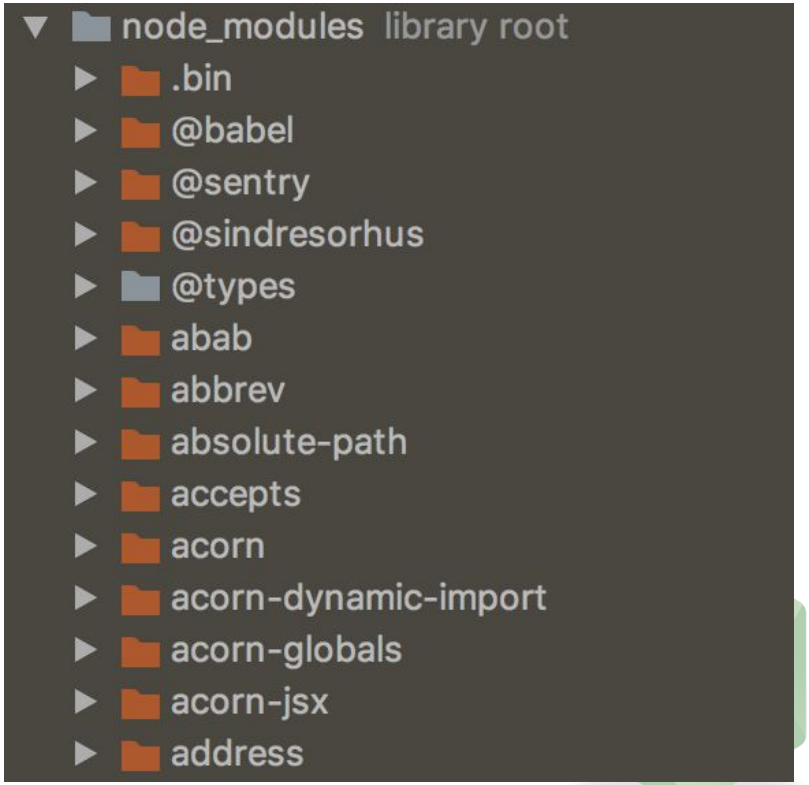
- Permite um **desenvolvimento 70% mais rápido**
    - **40% menos arquivos** e **33% menos LOC**
- Possibilita **2x mais requisições** com **⅛ do processamento** usado pelo Java
- **Carregamento** de página **35% mais rápido**



http://pieroxy.net/blog/2015/06/29/node_js_vs_java_nashorn.html

Pacotes/Módulos

- São **dependências** do **projeto**
  - Ficam no diretório "node_modules" da sua aplicação
- **Podem ser**
  - Nativos
  - De terceiros
  - Próprios



Node Package Manager (NPM)

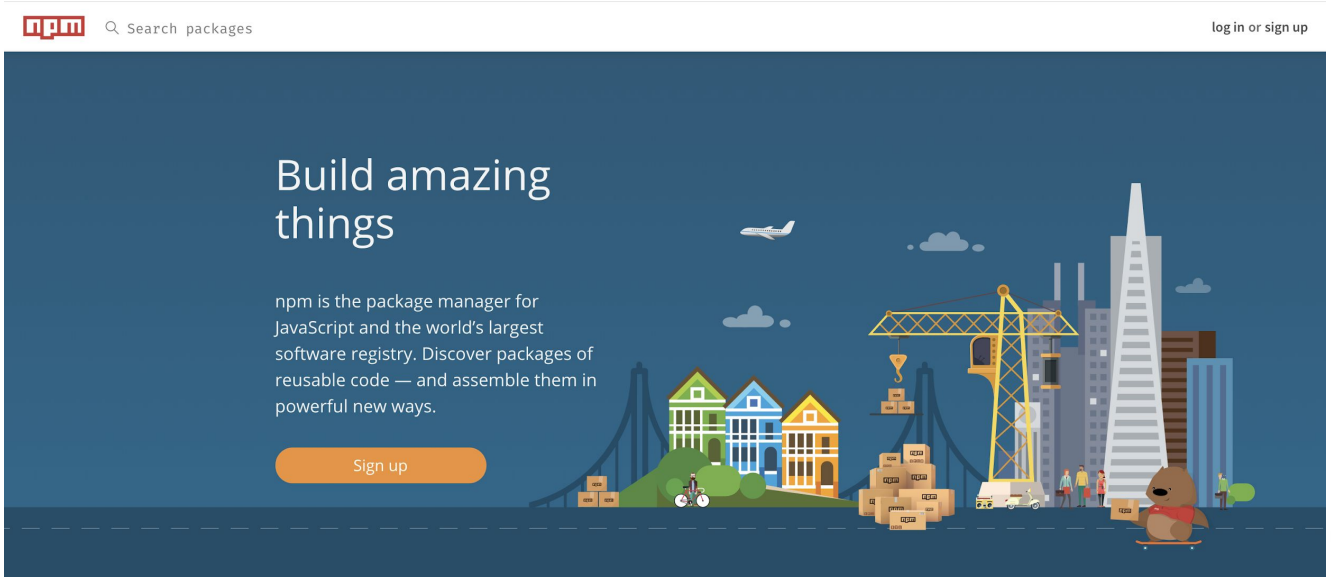
- **Gerenciador** de **pacotes** do **nodejs**
- Arquivo **package.json**
  - Define um módulo
  - Informa: nome, versão, endereço do repositório, dependências, scripts
- Comandos
  - Iniciar módulo: npm init <nome_pacote>
  - Instalar pacote: npm install <nome_pacote> --save (ou --save-dev)
  - Remover pacote: npm uninstall <nome_pacote>
  - Atualizar pacote: npm update <nome_pacote>

Módulos Nativos

- Escritos em
  - C, C++
  - Javascript
- Exemplo
  - Path
  - Url
  - Http
  - OS
  - Events
- https://nodejs.org/api

Módulos de terceiros

- Mais de 700.000 módulos publicados
- https://www.npmjs.com/



Módulos Próprios

- Declarando

```
//math.js
module.exports = {
   soma: function (a, b) { return a + b; }
}
```

```
//package.json

{
  "name": "math",
  "version": "1.0.0",
  "description": "",
  "main": "math.js",
}
```

- Consumindo

```
//outro.js
var math = require('./math');
math.soma(1, 2); //3
```

Hello World

- Criar arquivo chamado "server.js"

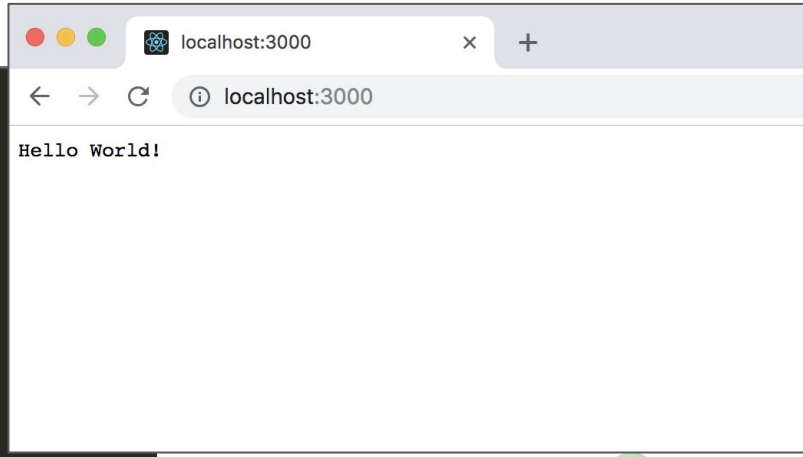
```
let http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

let server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log("Server running at http://"+hostname+":"+port);
});
```

- Rodar o comando "node server.js" no terminal

Hello World

- Criar arquivo chamado "server.js"

```
let http = require('http');

const hostname = '127.0.0.1';
const port = 3000;

let server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello World\n');
});

server.listen(port, hostname, () => {
  console.log("Server running at http://"+hostname+":"+port);
});
```



- Rodar o comando "node server.js" no terminal

Express

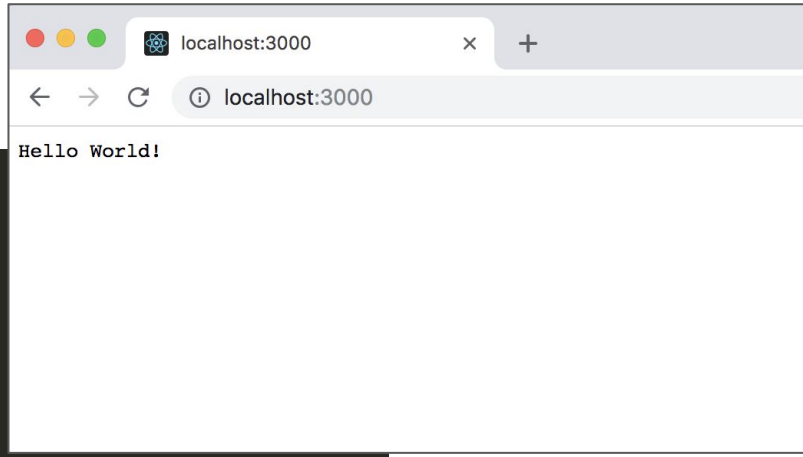
express

- O Express é um **framework** que **facilita criar aplicações web** utilizando **nodejs**.
- É um framework **rápido**, **minimalista** e **flexível** que fornece um **conjunto** robusto de **recursos** para **criar APIs** de **maneira fácil**.
- Escrito em JavaScript que **roda sobre o ambiente node.js** em tempo de execução.
- https://expressjs.com/pt-br/

Express: Hello World

- Criar arquivo chamado "server_express.js"

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(5000, function () {
  console.log('Example app listening on port 5000!');
});
```

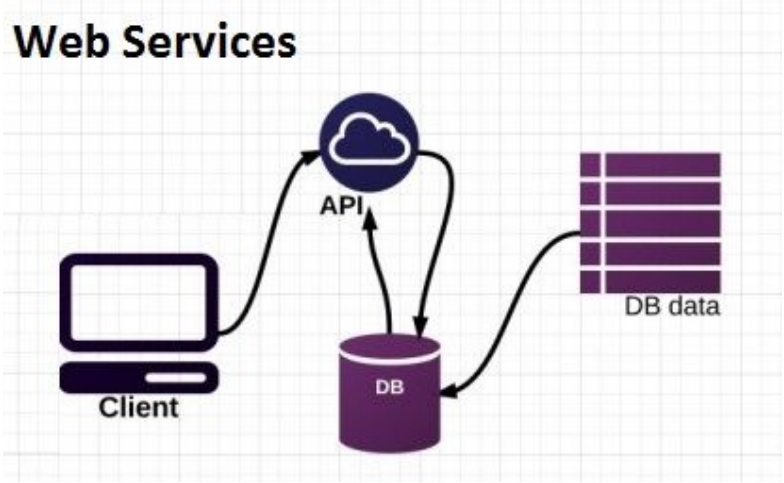


- Rodar o comando "**npm install express --save**"
- Rodar o comando "**node server_express.js**" no terminal

APIs REST

- **APIs** (Interface de Programação de Aplicações)
  - É um **conjunto de rotinas** e **padrões** estabelecidos por **um software** para a **utilização** das suas **funcionalidades** por **aplicativos** que **não pretendem envolver-se** em **detalhes** da **implementação** do **software**, mas **apenas usar seus serviços**.
- REST (Transferência de Estado Representacional)
  - É um **estilo de arquitetura** que **define** um conjunto **propriedades baseados** em **HTTP**.
  - **Padroniza URLs**, e **metadados** da **requisição**
  - **Diferencia** as **requisições** pelos verbos **HTTP**

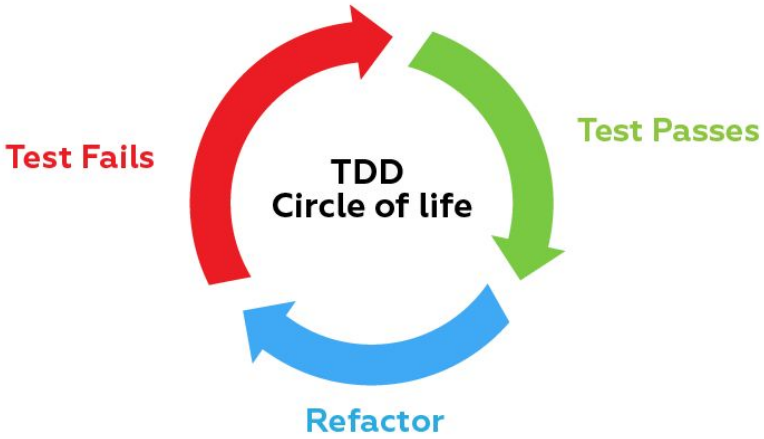
APIs REST



GET	/movies	Get list of movies
GET	/movies/:id	Find a movie by its ID
POST	/movies	Create a new movie
PUT	/movies	Update an existing movie
DELETE	/movies	Delete an existing movie

Test Driven Development (TDD)

- É uma **técnica** de **desenvolvimento** de **software** que se relaciona com o **conceito** de **validação**.
- Primeiramente o **desenvolvedor** escreve um **caso** de **teste automatizado** para uma **nova funcionalidade**.



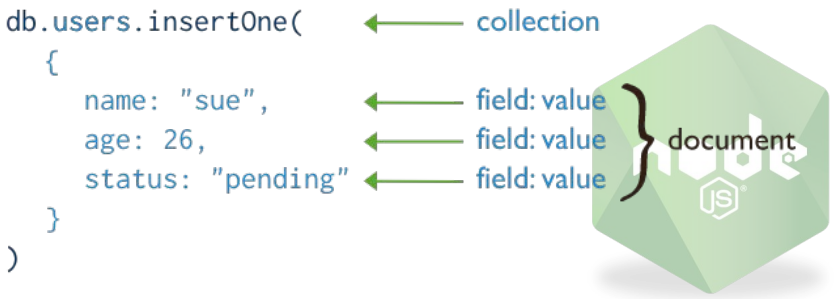
- Então, **produz** o **código** que **será validado** pelo **teste,** para posteriormente o **código ser refatorado.**
- Kent Beck, considerado o criador do TDD, declarou que o uso do **TDD encoraja** o **código simples** e **inspira confiança**.

Cloud Computing: Heroku

- Plataforma de **computação** em **nuvem** (PaaS), isso significa que você pode fazer **deploy sem** se **preocupar** com **configurações** de **hardware** e **SO**.
- Você só precisa se **focar na aplicação** e nos **componentes** de **infraestrutura** necessários (Banco de dados, cache, log, filas)
- Ele **suporta linguagens** por meio dos "**Buildpacks**"
    - Ruby, Python, Node, Php, Go, Java
- Os **aplicativos devem incluir** um arquivo "**Procfile**" que **especifica** aos **Dynos**, os **comandos** que deverão ser **executados**.

Mongodb

- É um software de **banco de dados orientado** a **documentos**
- De **código aberto**, **multiplataforma** e de **alta performance**
- Classificado como um programa de banco de dados **NoSQL**
- Usa **documentos semelhantes** a **JSON** com esquemas.
- Escrito na linguagem C++
    - O que o torna **portável** para **diferentes SOs**



Mongoose

mongoose

- **"Modelagem de objetos mongodb elegante para node.js"**
- O **mongoose foi criado** para:
  - **Facilitar** as **validações** do **MongoDB** no **nodejs**
  - **Fornecer** uma **solução** direta **baseada em esquemas** para **modelar** os **dados da aplicação**
  - **Construção** de **queries** de maneira simples

Mongoose Schema

- Tudo no **Mongoose** começa com um **esquema**.
- São **análogos** ao "**Modelo**" da arquitetura **MVC**.
- O **esquema mapeia** para **uma coleção** do **MongoDB** e **define a forma** que os **documentos** terão dentro **dessa coleção**.

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');

let Schema = mongoose.Schema;

let AnimalSchema = new Schema({
    name:  String
});
```

Operações com Mongoose

- **Operações**
  - find
  - findOne
  - findOneAndUpdate
  - count
  - remove
  - save

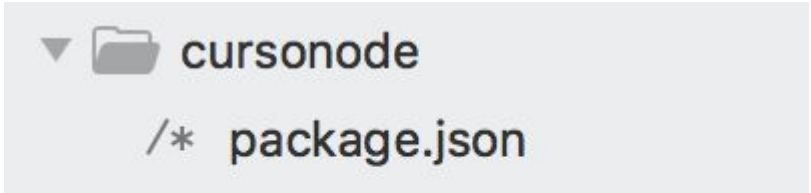
```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost/test');

let Schema = mongoose.Schema;

let AnimalSchema = new Schema({
    name:  String
});


//create
let dog = new AnimalSchema({
    name: "Cachorro"
});

dog.save();

// find

let search = AnimalSchema.find({name: "Cachorro"});
console.log(search); //prints Cachorro
```

Hands on: Workflow

- Criar novo projeto
- Criar o arquivo de entrada da aplicação
- Instalar dependências
- Criar o primeiro modelo
- Implementar rotas REST e testes automatizados para o CRUD
  - C - Create
  - R - Read
  - U - Update
  - D - Delete
- Rodar aplicação na nuvem
- Rodar testes automatizados na nuvem

Hands on: Criar um novo projeto

- Criar diretório "cursonode"
  - **mkdir cursonode**
- Navegar até a pasta
  - **cd cursonode**
- Inicializar o módulo npm
  - **npm init**
  - No "**entry point**" digitar **"server.js"**

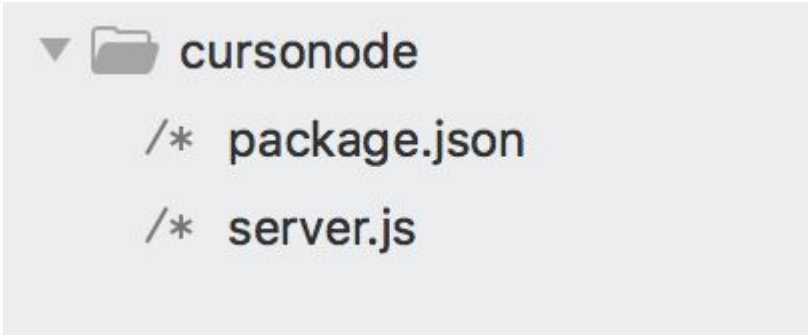



```
{
  "name": "cursonode",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Hands on: Criar o arquivo de entrada da aplicação

- Abra o projeto no editor de código

- Crie o arquivo "server.js" na raiz do projeto



Hands on: Instalando as dependências

- Vamos utilizar as seguintes dependências
  - Express, Body Parser, Mongoose
  - Mocha, Chai e Chai Http
- Para adicionar ...
  - npm install <nome_modulo> --save
  - Exemplo: npm install express --save
- Ou…
  - **npm install express body-parser mongoose mocha chai chai-http --save**

```
{
  "name": "cursonode",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.18.3",
    "chai": "^4.1.2",
    "chai-http": "^4.2.0",
    "express": "^4.16.3",
    "mocha": "^5.2.0",
    "mongoose": "^5.2.17"
  }
}
```

Hands on: Criar API Rest

- **API REST para lista de tarefas**
  - **GET** "/todo/:id?" busca um objeto
  - **PUT** "/todo/:id" atualiza um objeto
  - **POST** "/todo" cria um novo objeto
  - **DELETE** "/todo/:id" delete um objeto

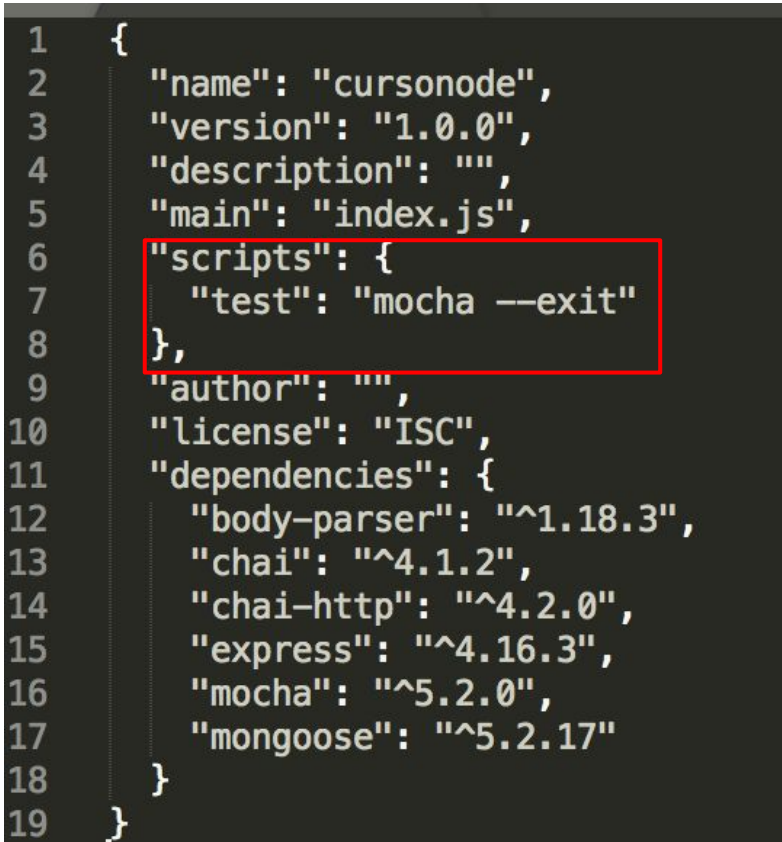
Hands on: Implementar rotas REST para o CRUD

- Importar dependências, implementar a rota default e executar o server

```
let express = require('express');
let app = express();

let bodyParser = require("body-parser");
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

app.get('/', function (req, res) {
  res.send('Hello World!'); //envia texto
});

//SERVER listening
let port = process.env.PORT || 5000;
app.listen(port, function () {
  console.log('Example app listening on port '+port);
});

//add exports to app at server.js
module.exports = app;
```

Hands on: Adicionando testes automatizados

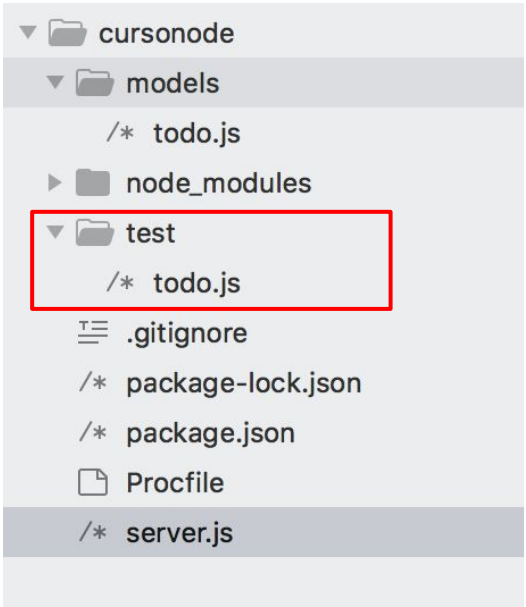
- Vamos utilizar Mocha e Chai para executar os testes automatizados
- Vamos adicionar o mocha dentro do script de teste no package.json



```
{
  "name": "cursonode",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "mocha --exit"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.18.3",
    "chai": "^4.1.2",
    "chai-http": "^4.2.0",
    "express": "^4.16.3",
    "mocha": "^5.2.0",
    "mongoose": "^5.2.17"
  }
}
```

Hands on: Adicionando testes automatizados

- Vamos criar o diretório "test" dentro da pasta raiz do projeto
- Vamos criar o arquivo "todo.js" dentro da pasta "test"



Hands on: Adicionando testes automatizados

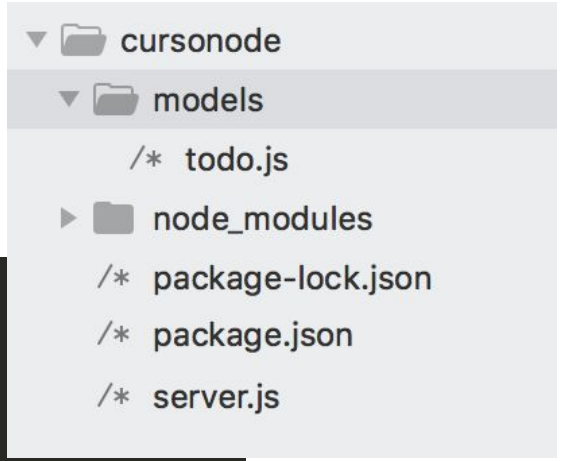
- Vamos inicializar nosso arquivo "todo.js" com as seguintes configurações e rodar o comando:
    - **npm test**

```
➜  cursonode git:(master) ✗ npm test

> cursonode@1.0.0 test /Users/renanpupin/workspace/cursonode
> mocha --exit



Example app listening on port 5000
  Test server working
    ✓ should get a server message


  1 passing (27ms)
```

```
let chai = require('chai');
let chaiHttp = require('chai-http');
chai.use(chaiHttp);
let expect = chai.expect;
let should = chai.should();

//carregando nossa API
let server = require('../server.js');

describe('Test server working', function(done) {
    it('should get a server message', function(done) {
        chai.request(server)
            .get('/')
            .end(function(err, res){
                expect(res.status).to.eql(200);
                done();
            });
    });
});
```

Hands on: Criando o primeiro modelo

- Vamos criar o modelo para a entidade "**ToDo**"
- **Crie** o **diretório "models" dentro** do **projeto**
- **Dentro** do **diretório** criado, **crie o arquivo "todo.js"**




```
let mongoose = require('mongoose');
let Schema = mongoose.Schema;

let schema = new Schema({
    title: {type: String, required: true},
    is_complete: {type: Boolean, default: false},
    created_at: {type: Date, required: true, default: Date.now},
    completed_at: {type: Date}
});

module.exports = mongoose.model('ToDo', schema);
```

Hands on: Implementar rotas REST para o CRUD

- Conectar no banco de dados (usar URL do banco de cada um) e importar modelo "ToDo"

```
6   app.use(bodyParser.urlencoded({ extended: false }));
7
8   let mongoose = require('mongoose');
9   mongoose.connect(
10      "<SUA_URL_DO_MLAB>",
11      { useNewUrlParser: true }
12  );
13
14  let ToDo = require("./models/todo");
15
16  app.get('/', function (req, res) {
```

Hands on: Adicionando testes automatizados

- Vamos criar os testes para as operações da nossa API REST
- GET

```
describe('Test ToDo is Working', function(done) {

    let id;

    //test GET
    it('should get ToDos', function(done) {
        chai.request(server)
            .get('/todo')
            .end(function(err, res){
                expect(res.body.success).to.eql(true);
                expect(res.body.todos).to.be.an('array');
                done();
            });
    });
});
```

Hands on: Implementar rotas REST para o CRUD

- Implementando o GET /todo

```
//get with callback
app.get('/todo', function (req, res) {
  ToDo
    .find()
    .exec((err, todos) => {
        if(!err){
            res.json({success: true, message: "ToDos buscados com sucesso.", todos });  //responde com um objeto json
        }else{
            res.json({success: false, message: err.message, todos: [] });
        }
    })
});
```

Hands on: Adicionando testes automatizados

- POST

```
describe('Test ToDo is Working', function(done) {

    let id;

    //test GET
    it('should get ToDos', function(done) {
        chai.request(server)
            .get('/todo')
            .end(function(err, res){
                expect(res.body.success).to.eql(true);
                expect(res.body.todos).to.be.an('array');
                done();
            });
    });

    //test POST
    it('should create ToDos', function(done) {
        chai.request(server)
            .post('/todo')
            .send({ title: "Todo Teste" })
            .end(function(err, res){
                expect(res.body.success).to.eql(true);
                expect(res.body.todo).to.not.be.undefined;
                expect(res.body.todo.is_complete).to.eql(false);

                id = res.body.todo._id; //salvando para alterar

                done();
            });
    });
});
```

Hands on: Implementar rotas REST para o CRUD

- Implementando o POST /todo

```
//post com async/await e try/catch
app.post('/todo', async(req, res) => {
    try{
        let title = req.body.title;

        let newTodo = new ToDo({
            title: title
        });

        let savedTodo = await newTodo.save();

        res.json({ success: true, message: "Successo!!!", todo: savedTodo });
    }catch(err){
        res.json({ success: false, message: err.message });
    }
});
```

Hands on: Adicionando testes automatizados

- PUT

```
//test PUT
it('should update ToDos', function(done) {
    chai.request(server)
        .put('/todo/'+id)
        .send({ is_complete: true })
        .end(function(err, res){
            expect(res.body.success).to.eql(true);
            expect(res.body.todo).to.not.be.undefined;
            expect(res.body.todo.is_complete).to.eql(true);

            done();
        });
});
```

Hands on: Implementar rotas REST para o CRUD

- Implementando o PUT /todo:id

```
//put
app.put('/todo/:id', async(req, res) => {
    try{
        let id = req.params.id;
        let is_complete = req.body.is_complete;

        let todo = await ToDo.findOne({_id: id});
        console.log(todo);

        if(is_complete !== undefined){
            todo.is_complete = is_complete;
            todo.completed_at = new Date();
        }

        let updatedTodo = await todo.save();
        console.log("updatedTodo", updatedTodo);

        res.json({ success: true, message: "Successo!!!", todo: updatedTodo });
    }catch(err){
        res.json({ success: false, message: err.message });
    }
});
```

Hands on: Adicionando testes automatizados

- DELETE

```
//test DELETE
it('should remove ToDos', function(done) {
    chai.request(server)
        .delete('/todo/'+id)
        .end(function(err, res){
            expect(res.body.success).to.eql(true);

            done();
        });
});
```

Hands on: Implementar rotas REST para o CRUD

- Implementando o DELETE /todo:id

```
//delete
app.delete('/todo/:id', async(req, res) => {
    try{
        let id = req.params.id;

        let todo = await ToDo.findOne({_id: id});
        console.log(todo);

        await todo.remove();

        res.json({ success: true, message: "Successo!!!" });
    }catch(err){
        res.json({ success: false, message: err.message });
    }
});
```

Hands on: Adicionando testes automatizados

Hands on: Subindo o código para o Git

- Criar Procfile na raiz do projeto

```
//Procfile
web: node server.js
```

- Criar arquivo ".gitignore" na raiz do projeto e adicionar "node_modules

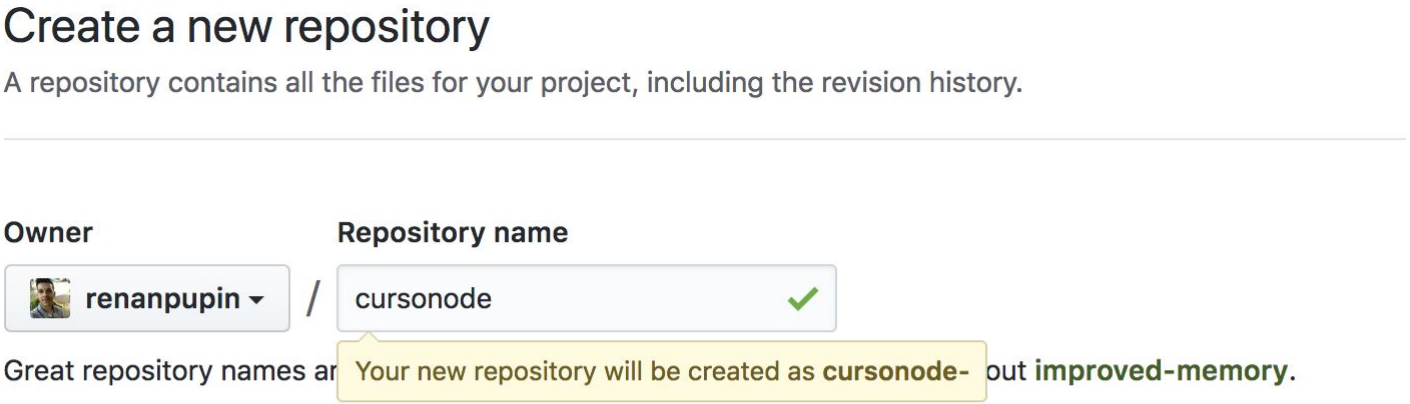
```
//.gitignore
node_modules
```

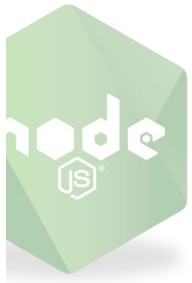
Hands on: Subindo o código para o Git

- Inicializar um novo repositório Git

```
➜  cursonode2 git init
Initialized empty Git repository in /Users/renanpupin/workspace/cursonode2/.git/
```

- Criar repositório chamado "cursonode" no Github



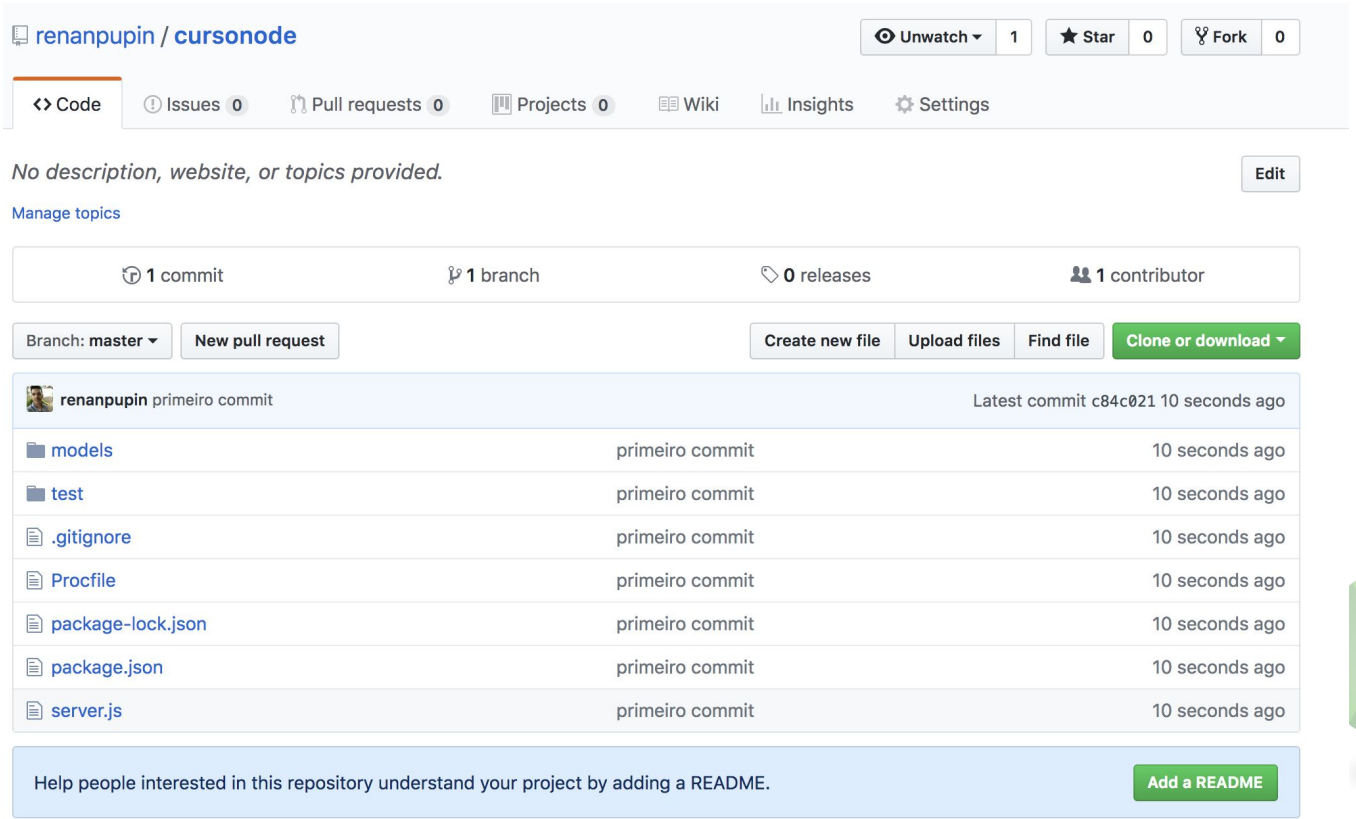


Hands on: Subindo o código para o Git

- Setar a origem do repositório para o Github
  - **git remote add origin <meu_repositorio.git>**

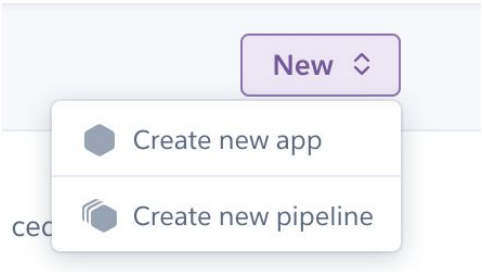
- Subir o código para o git
  - **git add .**
  - **git commit -m "primeiro commit"**
  - **git push origin master**

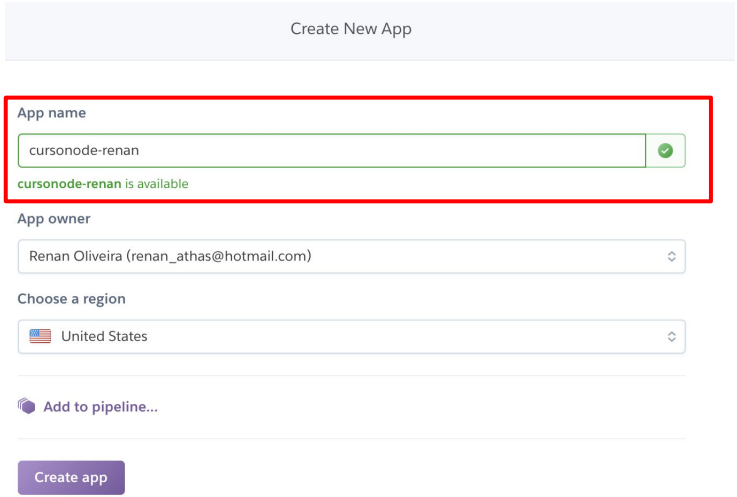
Hands on: Subindo o código para o Git

- 

Hands on: Rodando na nuvem

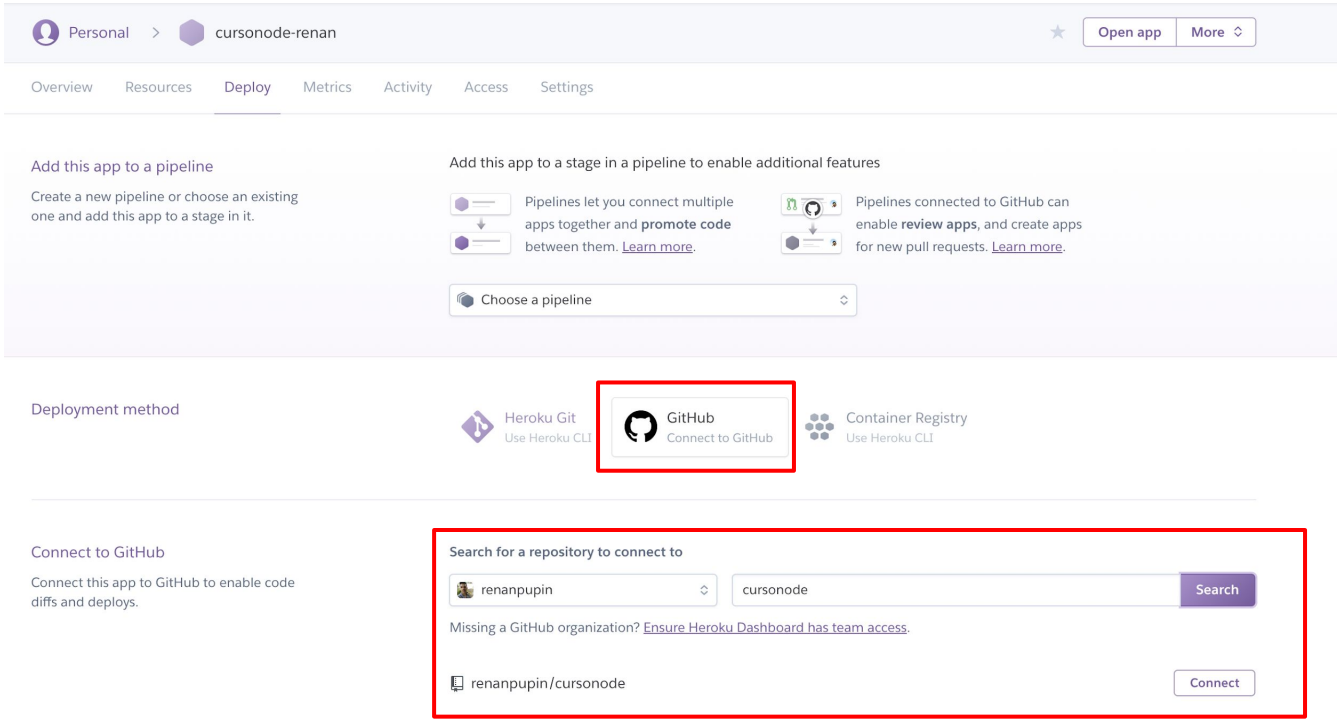
- Após fazer cadastro na plataforma Heroku, abra o link:
  - https://dashboard.heroku.com/apps
- Crie um novo app clicando botão do canto direito da tela





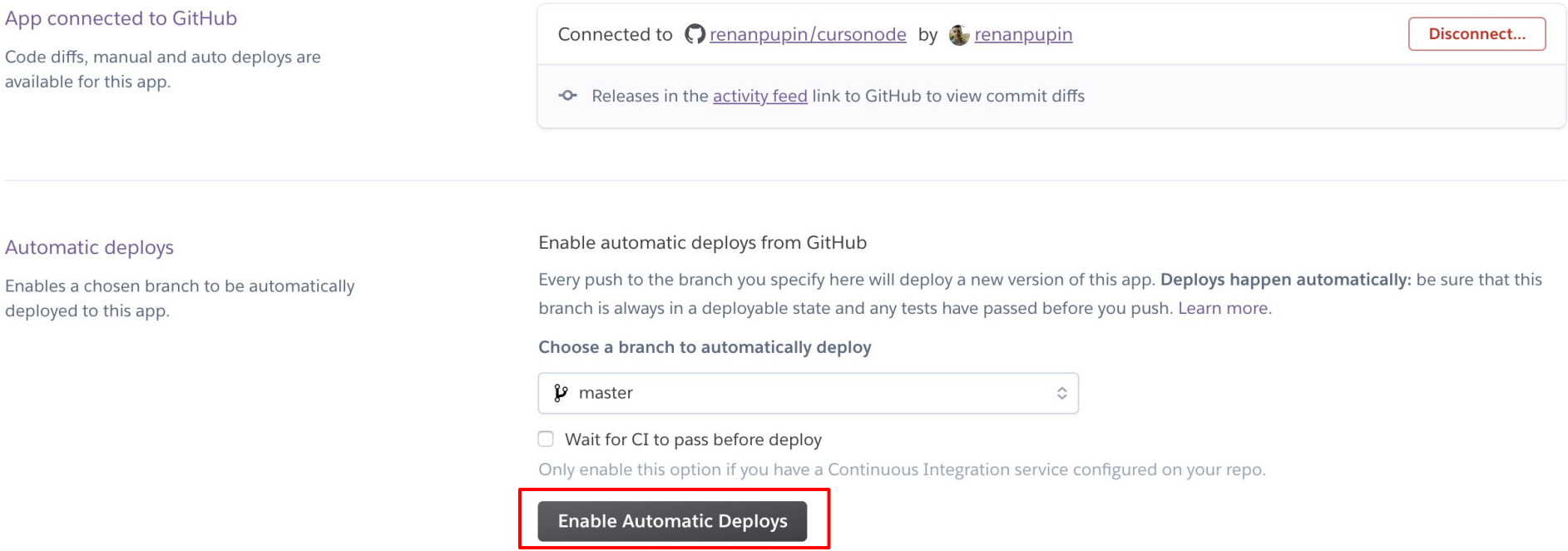
Hands on: Rodando na nuvem

- Vamos conectar o Heroku ao Github



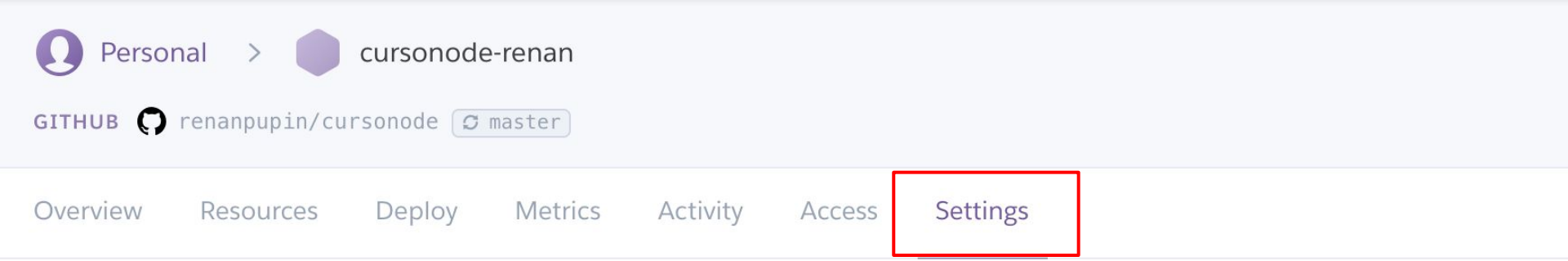
Hands on: Rodando na nuvem

- Após conectar vamos habilitar os deploys automáticos



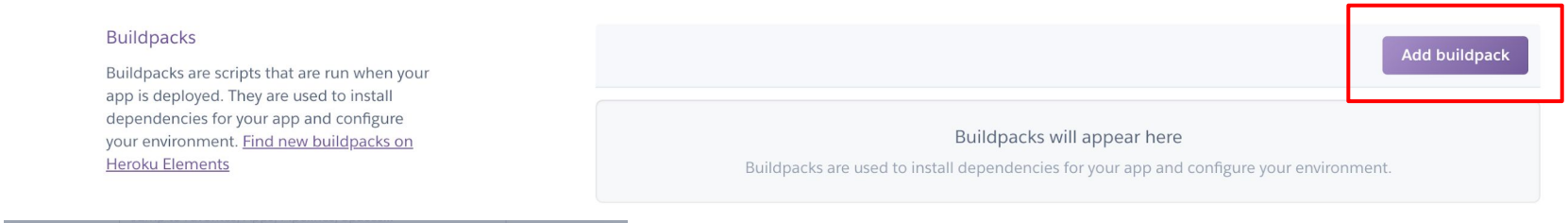
Hands on: Rodando na nuvem

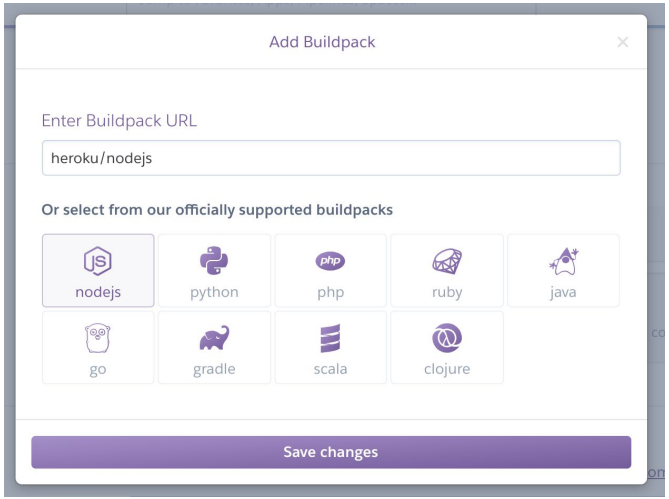
- Vamos entrar na aba settings

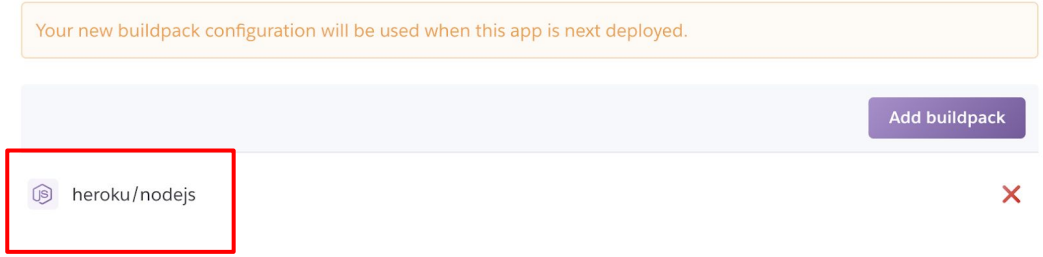


Hands on: Rodando na nuvem

- Adicionar o buildpack do nodejs







Hands on: Rodando na nuvem

- Na aba deploy, vamos fazer um novo deploy manual para ativar o buildpack

Manual deploy

Deploy the current state of a branch to this app.

Deploy a GitHub branch

This will deploy the current state of the branch you specify below. Learn more.

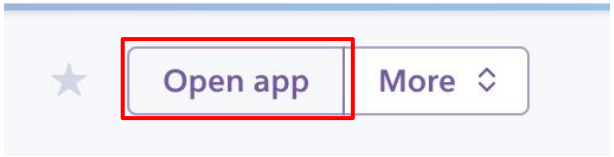
Choose a branch to automatically deploy

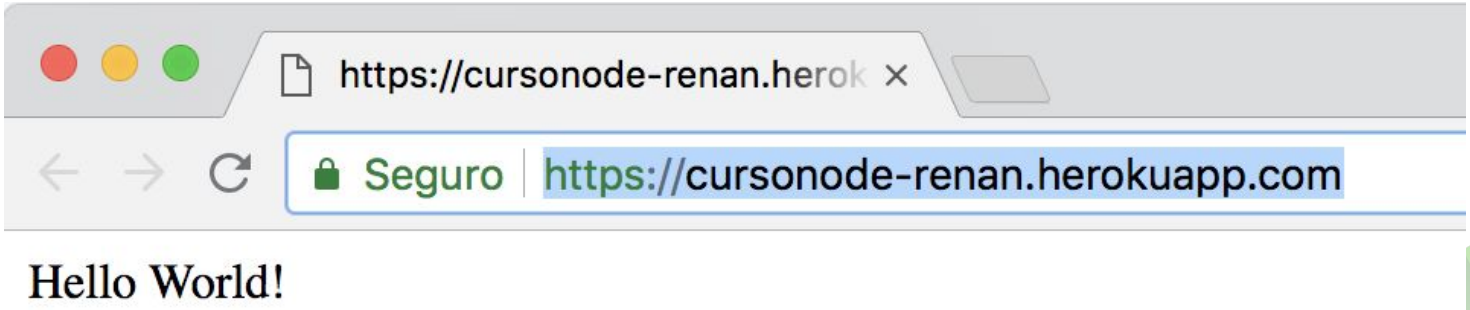
master

Deploy Branch

Hands on: Rodando na nuvem

- Se abrirmos a aplicação pelo botão "Open App", podemos ver nossa aplicação Hello World.





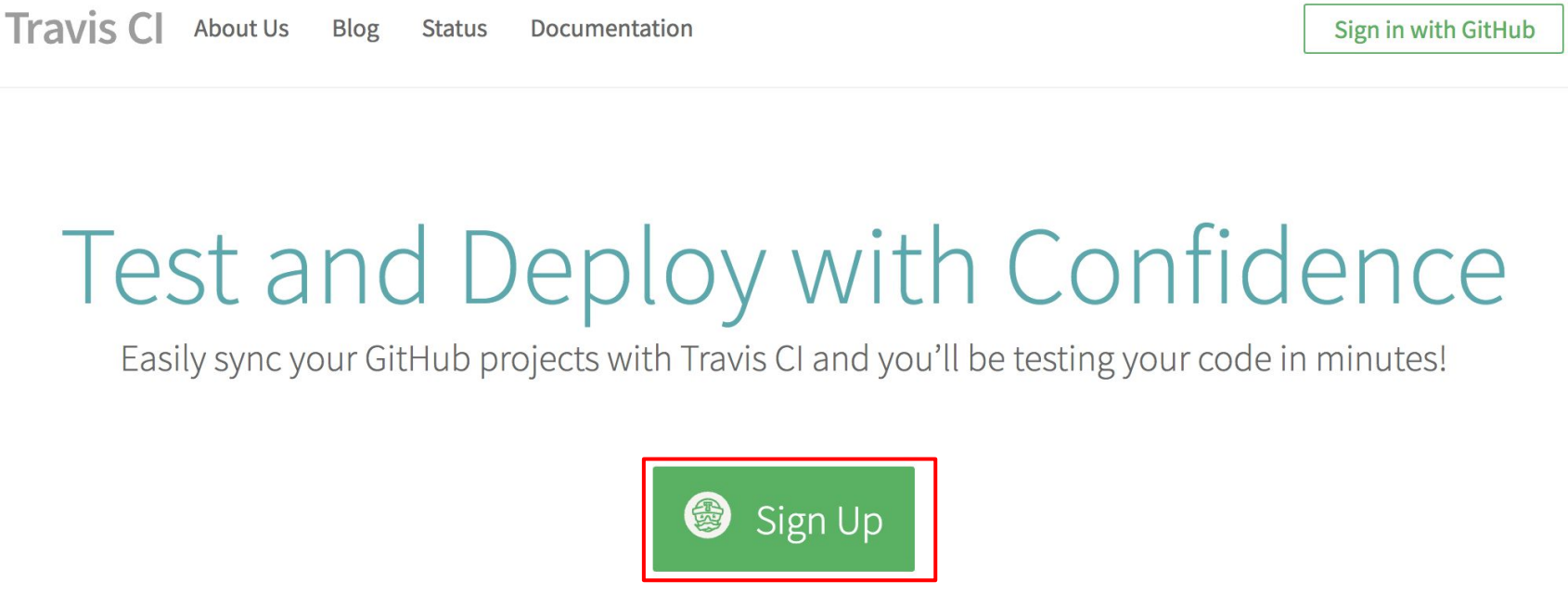
Hands on: Rodando testes automatizados no CI

- Criar arquivo ".travis.yml" e **subir** para o **git**
  - **git add .**
  - **git commit -m "add travis.yml"**
  - **git push origin master**

```
language: node_js
node_js:
  - "9.10.0"
services:
  - mongodb
env:
- NODE_ENV=TEST
```

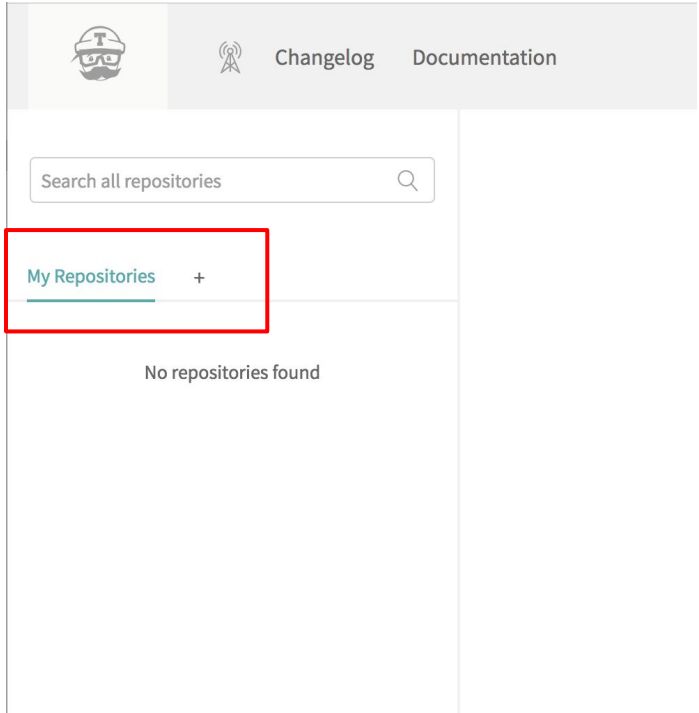
Hands on: Rodando testes automatizados no CI

- Conectar no site do TravisCI



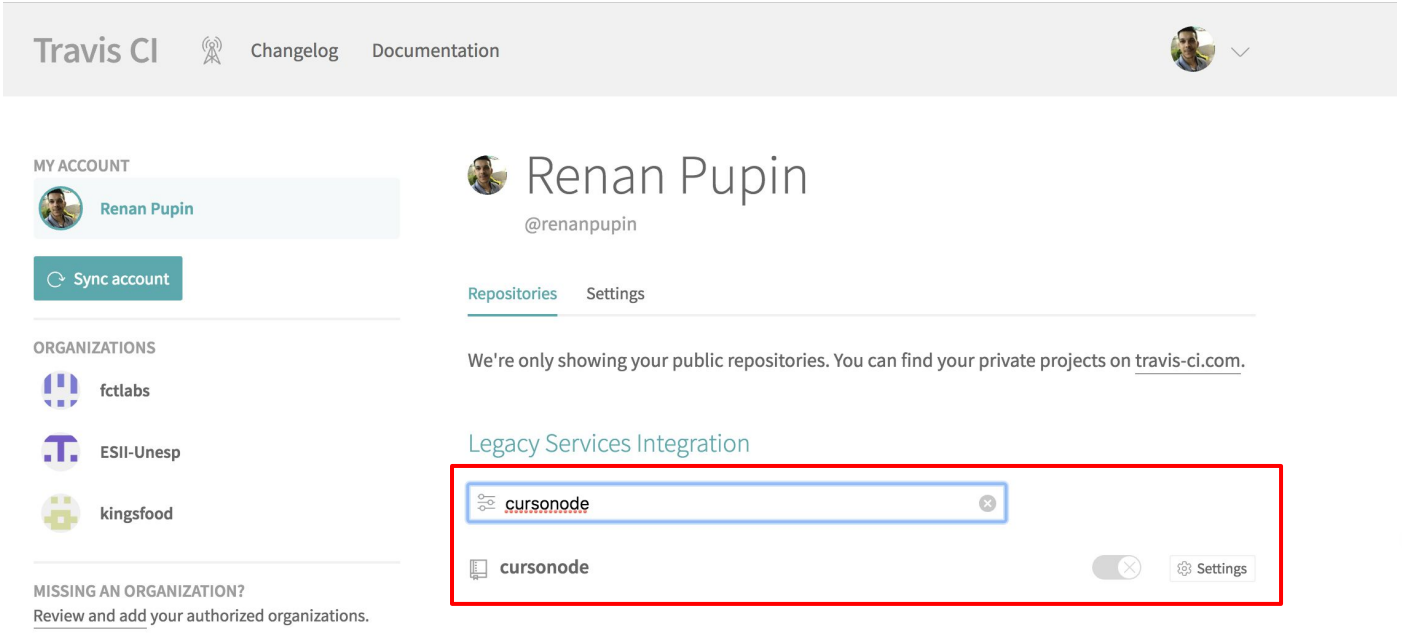
Hands on: Rodando testes automatizados no CI

- Vamos conectar o github no TravisCI



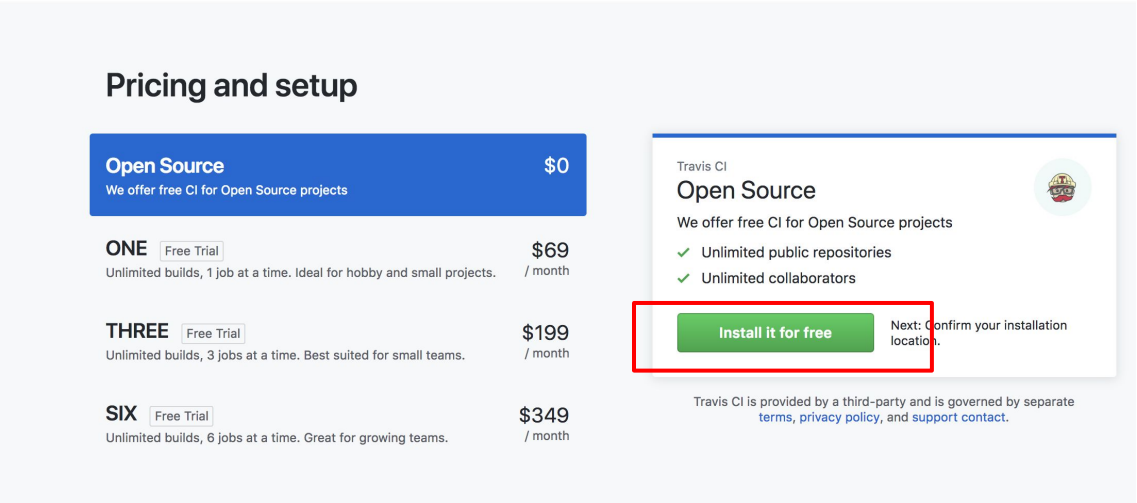
Hands on: Rodando testes automatizados no CI

- Selecione o repositório e habilite



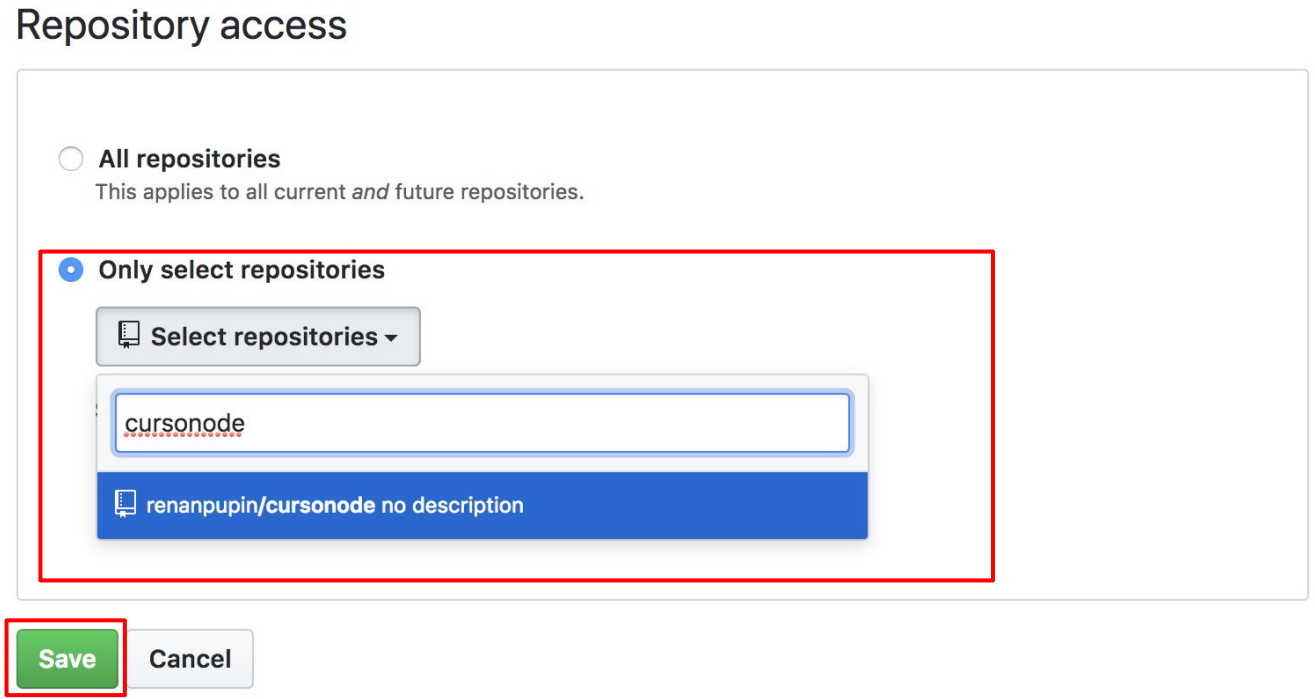
Hands on: Rodando testes automatizados no CI

- Vamos adicionar o app do travis no github
  - https://github.com/marketplace/travis-ci/
- Selecione o plano opensource e pressione o botão "Install it for free"



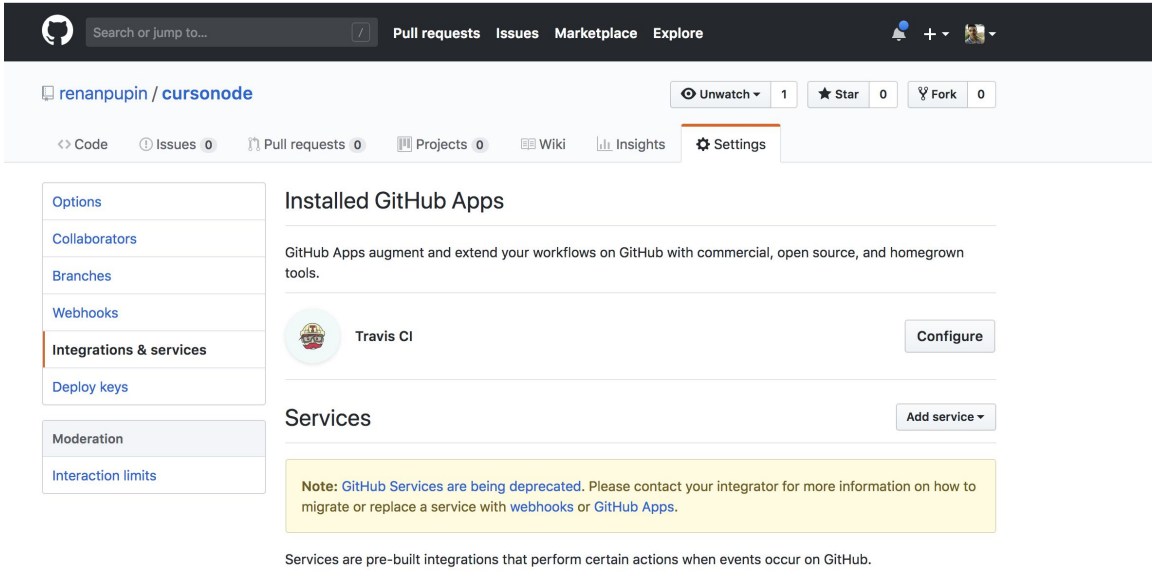
Hands on: Rodando testes automatizados no CI

- Selecione o repositório e pressione "Save"



Hands on: Rodando testes automatizados no CI

- Agora o Github já possui o app do travis instalado

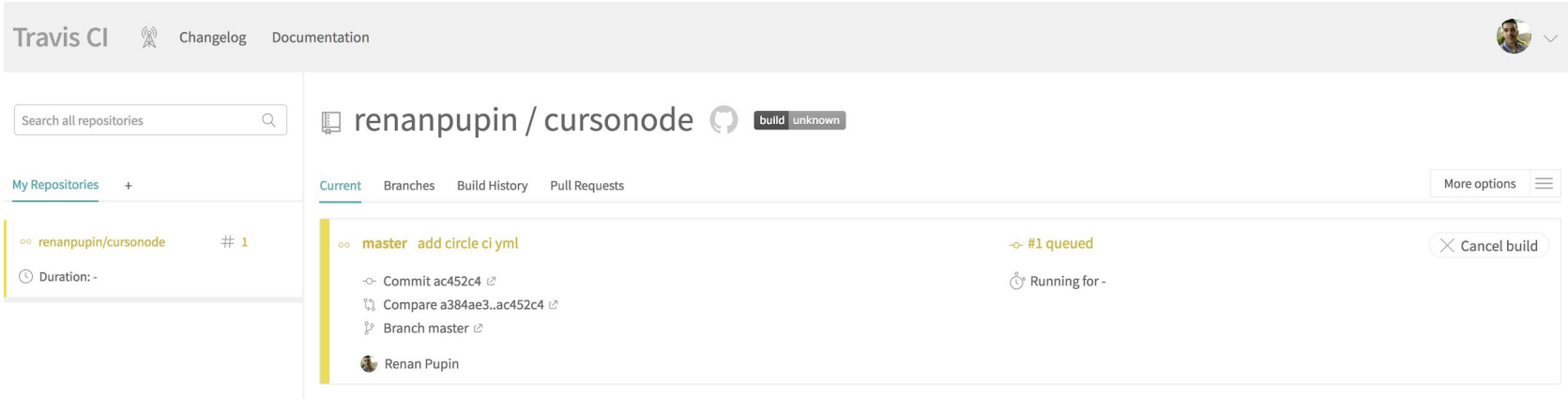


Hands on: Rodando testes automatizados no CI

- No Heroku, vamos habilitar a opção de apenas fazer deploy quando passar nos testes.
- Na aba deploy, marque a opção "Wait for CI to pass before deploy"

Hands on: Rodando testes automatizados no CI

- Vamos fazer um novo commit no git para fazer com o que o Travis execute um novo build ao receber um novo push
- Altere a resposta "Hello World!" para "Hello World Travis!" no server.js
    - **res.send('Hello World Travis!');**



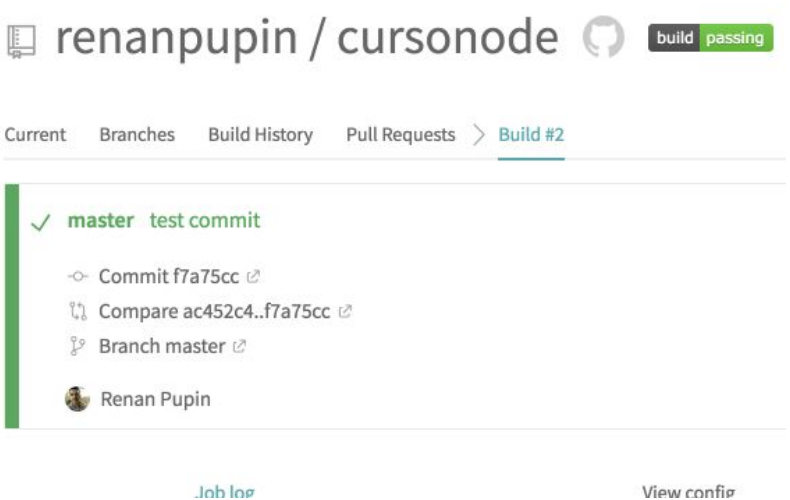
Hands on: Rodando testes automatizados no CI

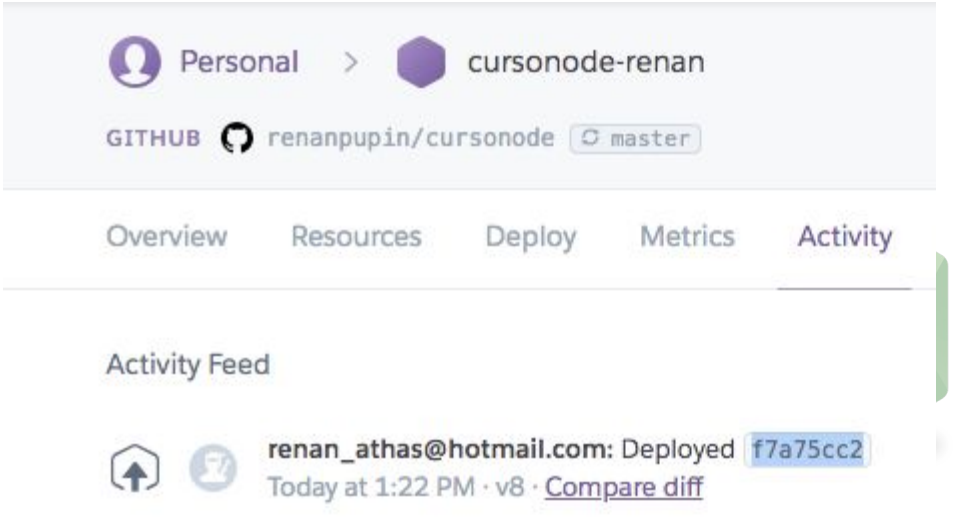
- Em seguida após rodar os testes, o Travis irá completar o build

```
> cursonode@1.0.0 test /home/travis/build/renanpupin/cursonode
> mocha --exit



Example app listening on port 5000
  Test server working
    ✓ should get a server message

  Test ToDo is Working
    ✓ should get ToDos (132ms)
    ✓ should create ToDos
{ is_complete: false,
  _id: 5bae50e83390a10aa1d3f52c,
  title: 'Todo Teste',
  created_at: 2018-09-28T16:03:52.008Z,
  __v: 0 }
updatedTodo { is_complete: true,
  _id: 5bae50e83390a10aa1d3f52c,
  title: 'Todo Teste',
  created_at: 2018-09-28T16:03:52.008Z,
  __v: 0,
  completed_at: 2018-09-28T16:03:52.035Z }
    ✓ should update ToDos
{ is_complete: true,
  _id: 5bae50e83390a10aa1d3f52c,
  title: 'Todo Teste',
  created_at: 2018-09-28T16:03:52.008Z,
  __v: 0,
  completed_at: 2018-09-28T16:03:52.035Z }
    ✓ should remove ToDos


  5 passing (233ms)
```

Hands on: Rodando testes automatizados no CI

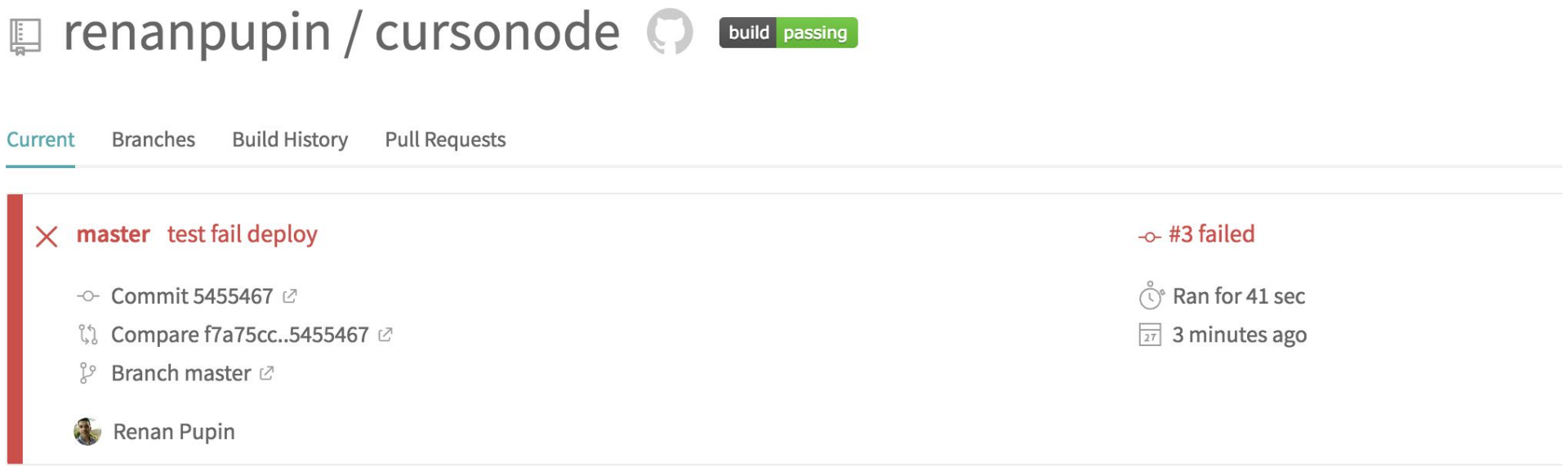
- Após fazer o build, em caso de sucesso nossa aplicação no heroku será atualizada





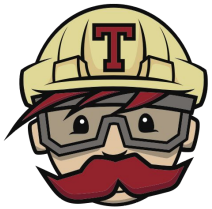
Hands on: Rodando testes automatizados no CI

- Se forçarmos um erro no teste, o teste automatizado não irá funcionar e a aplicação não será publicada automaticamente



Hands on: Bônus?

- Testes no FalaFreud



Dúvidas?

Links Úteis

- Código Fonte
  - https://github.com/nodejs/node
- Documentação
  - https://nodejs.org/en/docs/
- API
  - https://nodejs.org/api/
- NodeSchool
  - https://nodeschool.io/pt-br/
- Stack Overflow
  - https://stackoverflow.com/questions/tagged/node.js
- Npm
  - https://www.npmjs.com/

Comunidade

- We encourage all kinds of contribution from the community.
- The Node.js community is large, inclusive, and excited to enable as many users to contribute in whatever way they can. If you want to report an issue, help with documentation or contribute to the code base of the project, you've come to the right place. Explore our community resources to find out how you can help:

- https://nodejs.org/en/about/community/

Referências

- https://www.slideshare.net/giovanni.bassi/introduo-ao-nodejs-37802907?qid=71b312be-b14b-424c-b6c7-34fba93f5e78&v=&b=&from_search=4
- https://www.slideshare.net/search/slideshow?searchfrom=header&q=nodejs
- https://www.netguru.co/blog/pros-cons-use-node.js-backend
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Reference/Lexical_grammar
- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Values,_variables,_and_literals

Referências

- https://developer.mozilla.org/pt-BR/docs/Web/JavaScript/Guide/Closures

Obrigado!

- @renanpupin

  renan@falafreud.com

- @andreoandreotti

  andre@falafreud.com