

[Open in app](#)

Rafael Berçam

187 Followers

[About](#)[Follow](#)

You can now subscribe to get stories delivered directly to your inbox.

[Got it](#)

Automação de Testes API com HTTParty e Cucumber (BDD)



Rafael Berçam Jul 18, 2018 · 10 min read

Makes http fun again!



Testes em API com Cucumber e HTTParty

Rafael Berçam | Analista QA Automatizado

O HTTParty, é uma Ruby gem para realizar requisições de *web services* e examinar as saídas resultantes dessas requisições. Por padrão, ele gerará a resposta (resultado produzido) como um objeto Ruby no formato *pp* (*pretty print* que é útil para aumentar a estrutura da saída). Isso também pode ser substituído para saídas do tipo XML ou JSON.

[Open in app](#)

Pré requisitos

Para termos acesso aos métodos dessa Ruby gem precisamos instalar através do terminal utilizando o seguinte comando:

```
gem install httparty
```

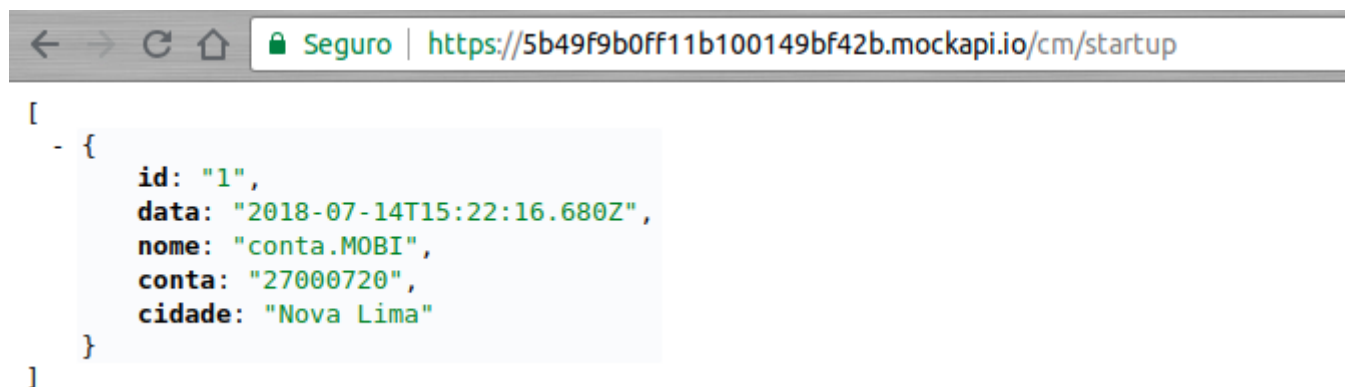
Lembrando que sempre é bom conferir a versão mais recente no site [rubygems](https://rubygems.org/) e copiar a versão mais recente para seu Gemfile que veremos mais à frente.

Teste com HTTParty

Nesse pequeno bloco vamos realizar um pequeno teste usando o HTTParty e ver como ele realiza as requisições.

Para realizar nossos testes foi utilizado o site <https://www.mockapi.io> onde vou deixar disponível esta rota para realizar os testes

<https://5b49f9b0ff11b100149bf42b.mockapi.io/cm/startup>



vamos realizar um primeiro teste realizar uma consulta usando o método `'get'` nessa rota para ver o que o HTTParty consegue executar.

[Open in app](#)

```
5 puts "response headers: #{response.headers}"
6 puts "response body: #{response.body}"
7
8
```

```
bercam@bercam-RV411:~/Documentos/HTTParty$ ruby HTTParty.rb
response code :200
response message: OK
response headers: {"server"=>["Cowboy"], "connection"=>["close"], "x-powered-by"=>["Express"], "access-control-allow-origin"=>["*"], "access-control-allow-methods"=>["GET,PUT,POST,DELETE,OPTIONS"], "access-control-allow-headers"=>["X-Requested-With,Content-Type,Cache-Control,access token"], "content-type"=>["application/json"], "content-length"=>["106"], "etag"=>["1688021665"], "vary"=>["Accept-Encoding"], "date"=>["Sun, 15 Jul 2018 13:12:06 GMT"], "via"=>["1.1 vegur"]}
response body: [{"id":"1","data":"2018-07-14T15:22:16.680Z","nome":"conta.MOBI","conta":"27000720","cidade":"Nova Lima"}]
bercam@bercam-RV411:~/Documentos/HTTParty$
```

Teste de execução da consulta

Para realizar este teste eu criei um arquivo com extensão `.rb` e executei o comando “`ruby <nome_arquivo>.rb`”.

Como podemos ver no resultado da requisição o HTTParty consegue consumir a requisição na API tendo acesso à atributos como

- code response
- message
- headers
- body

Vamos realizar agora um teste para cadastrar um novo registro utilizando o método ‘*post*’ para vermos como se dá essa estrutura.

```
HTTParty.rb
1 require 'httparty'
2
3 response = HTTParty.post('http://5b49f9b0ff11b100149bf42b.mockapi.io/cm/startup',
4   :body => {"nome":"symla","cidade":"Belo Horizonte"})
5 puts "response code :#{response.code}"
6 puts "response message: #{response.message}"
7 puts "response headers: #{response.headers.inspect}"
8 puts "response body: #{response.body}"
```

```
bercam@bercam-RV411:~/Documentos/HTTParty$ ruby HTTParty.rb
response code :201
response message: Created
response headers: {"server"=>["Cowboy"], "connection"=>["close"], "x-powered-by"=>["Express"], "access-control-allow-origin"=>["*"], "access-control-allow-methods"=>["GET,PUT,POST,DELETE,OPTIONS"], "access-control-allow-headers"=>["X-Requested-With,Content-Type,Cache-Control,access token"], "content-type"=>["application/json"], "content-length"=>["105"], "vary"=>["Accept-Encoding"], "date"=>["Sun, 15 Jul 2018 13:35:43 GMT"], "via"=>["1.1 vegur"]}
response body: {"id":"2","data":"2018-07-15T09:19:24.577Z","nome":"symla","conta":"21421999","cidade":"Belo Horizonte"}
bercam@bercam-RV411:~/Documentos/HTTParty$
```

Teste da requisição de cadastro

[Open in app](#)

cidade no formato json.

Agora se acessar o endereço da API vamos ver os dois registros cadastrados



```
[
  - {
    id: "1",
    data: "2018-07-14T15:22:16.680Z",
    nome: "conta.MOBI",
    conta: "27000720",
    cidade: "Nova Lima"
  },
  - {
    id: "2",
    data: "2018-07-15T09:19:24.577Z",
    nome: "sympia",
    conta: "21421999",
    cidade: "Belo Horizonte"
  }
]
```

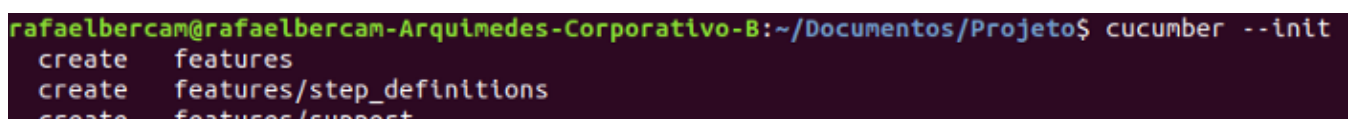
Super divertido certo? Mas ainda estamos longe de ter uma estrutura legal para realizar os testes nessa API, e com isso no próximo tópico vou mostrar como integrar o HTTParty com o Cucumber utilizando de padrões e boas práticas para obtermos um teste automatizado com baixo acoplamento.

Sugiro a usar um editor de texto como o Visual Studio Code ou um outro de sua preferência.

Projeto: Estrutura e Configuração

Crie a pasta [Projeto] e pelo terminal dentro desta pasta vamos digitar o comando :

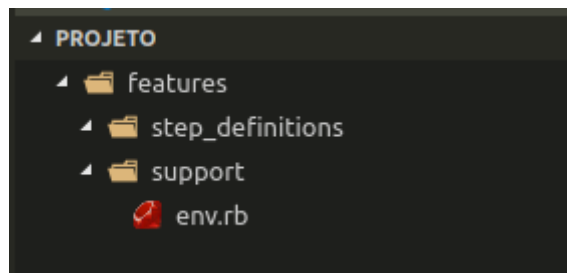
```
cucumber --init
```



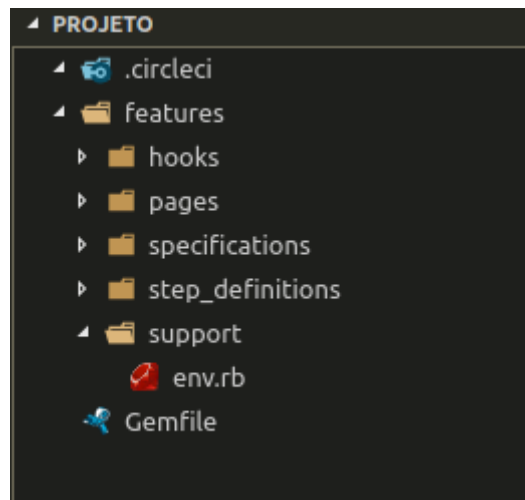
```
rafaelbercam@rafaelbercam-Arquimedes-Corporativo-B:~/Documentos/Projeto$ cucumber --init
create  features
create  features/step_definitions
create  features/support
```

[Open in app](#)

Veja que o Cucumber cria sua estrutura



Agora vamos dar uma incrementada nessa organização de diretórios dentro no nosso projeto

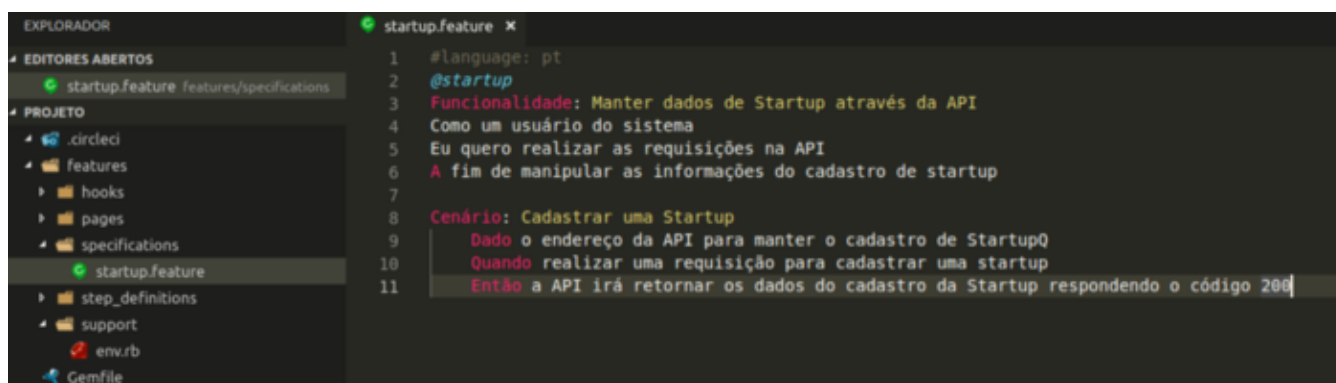


Estrutura de pastas sugerida

Além das pastas `step_definitions` e `support` criadas pelo Cucumber, eu criei outras 3 pastas dentro do diretório `[features]` que são:

- **[hooks]** -> Nesta pasta vamos guardar os arquivos responsáveis por instanciar nossas variáveis a cada chamada do cucumber
- **[specifications]** -> Nesta pasta serão salvos os arquivos `.features`
- **[pages]** -> Nesta pasta serão salvos os arquivos das classes do HTTParty contendo as requisições.

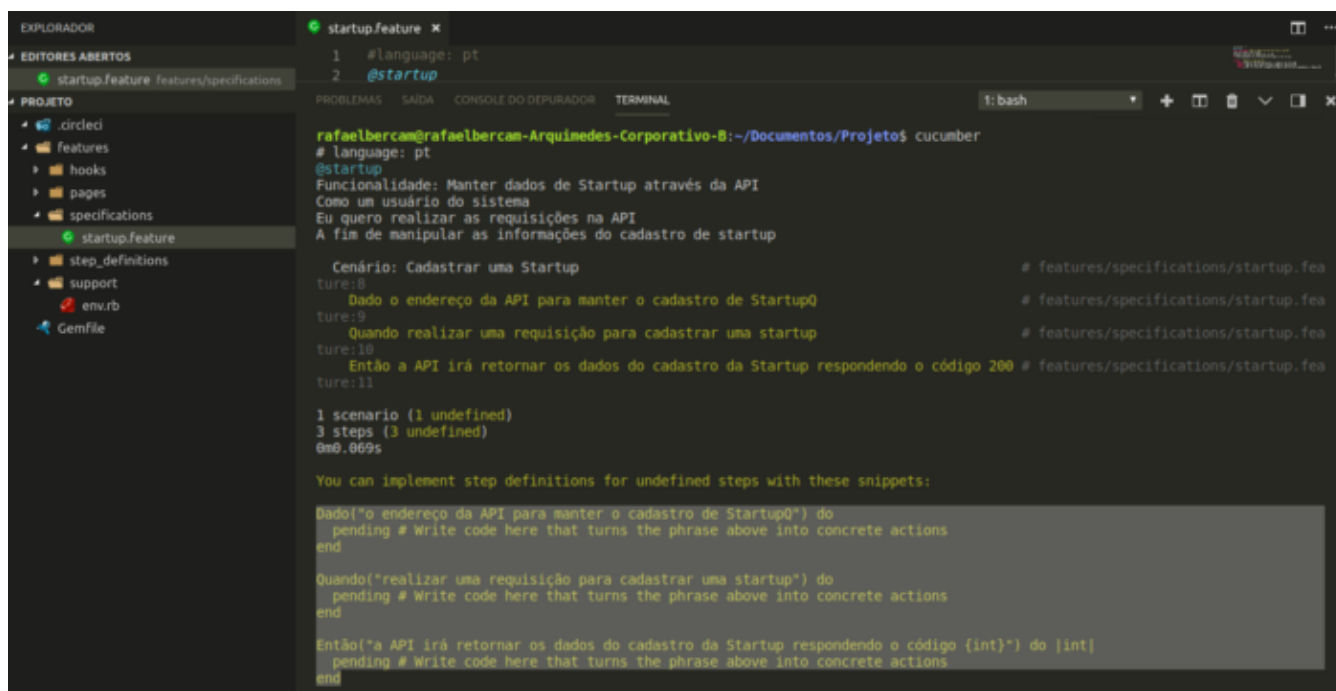
E uma pasta na raiz do projeto com nome `[.circleci]` que vamos usar para integrar nosso projeto no **CircleCI** que veremos mais adiante.

[Open in app](#)

```
1 #language: pt
2 @startup
3 Funcionalidade: Manter dados de Startup através da API
4 Como um usuário do sistema
5 Eu quero realizar as requisições na API
6 A fim de manipular as informações do cadastro de startup
7
8 Cenário: Cadastrar uma Startup
9     Dado o endereço da API para manter o cadastro de StartupQ
10    Quando realizar uma requisição para cadastrar uma startup
11    Então a API irá retornar os dados do cadastro da Startup respondendo o código 200
```

Vamos escrever uma pequena feature para cadastrar uma startup na nossa API.

Após redigir vamos na pasta raiz do projeto e digitar “cucumber” como no exemplo abaixo.



```
rafaelbercam@rafaelbercam-Arquimedes-Corporativo-B:~/Documentos/Projeto$ cucumber
# language: pt
@startup
Funcionalidade: Manter dados de Startup através da API
Como um usuário do sistema
Eu quero realizar as requisições na API
A fim de manipular as informações do cadastro de startup

Cenário: Cadastrar uma Startup # features/specifications/startup.feature:8
  Dado o endereço da API para manter o cadastro de StartupQ # features/specifications/startup.feature:9
  Quando realizar uma requisição para cadastrar uma startup # features/specifications/startup.feature:10
  Então a API irá retornar os dados do cadastro da Startup respondendo o código 200 # features/specifications/startup.feature:11

1 scenario (1 undefined)
3 steps (3 undefined)
0m0.069s

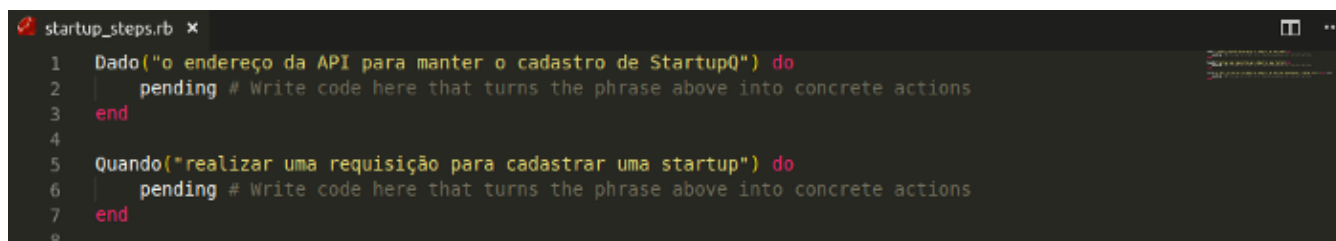
You can implement step definitions for undefined steps with these snippets:

Dado("o endereço da API para manter o cadastro de StartupQ") do
  pending # Write code here that turns the phrase above into concrete actions
end

Quando("realizar uma requisição para cadastrar uma startup") do
  pending # Write code here that turns the phrase above into concrete actions
end

Então("a API irá retornar os dados do cadastro da Startup respondendo o código {int}") do |int|
  pending # Write code here that turns the phrase above into concrete actions
end
```

Como não temos os passos implementados o cucumber gera o código para podemos implementar no nosso projeto. Copie a parte selecionada acima e vamos criar dentro do diretório ‘features/step_definitions’ um arquivo **startup_steps.rb**



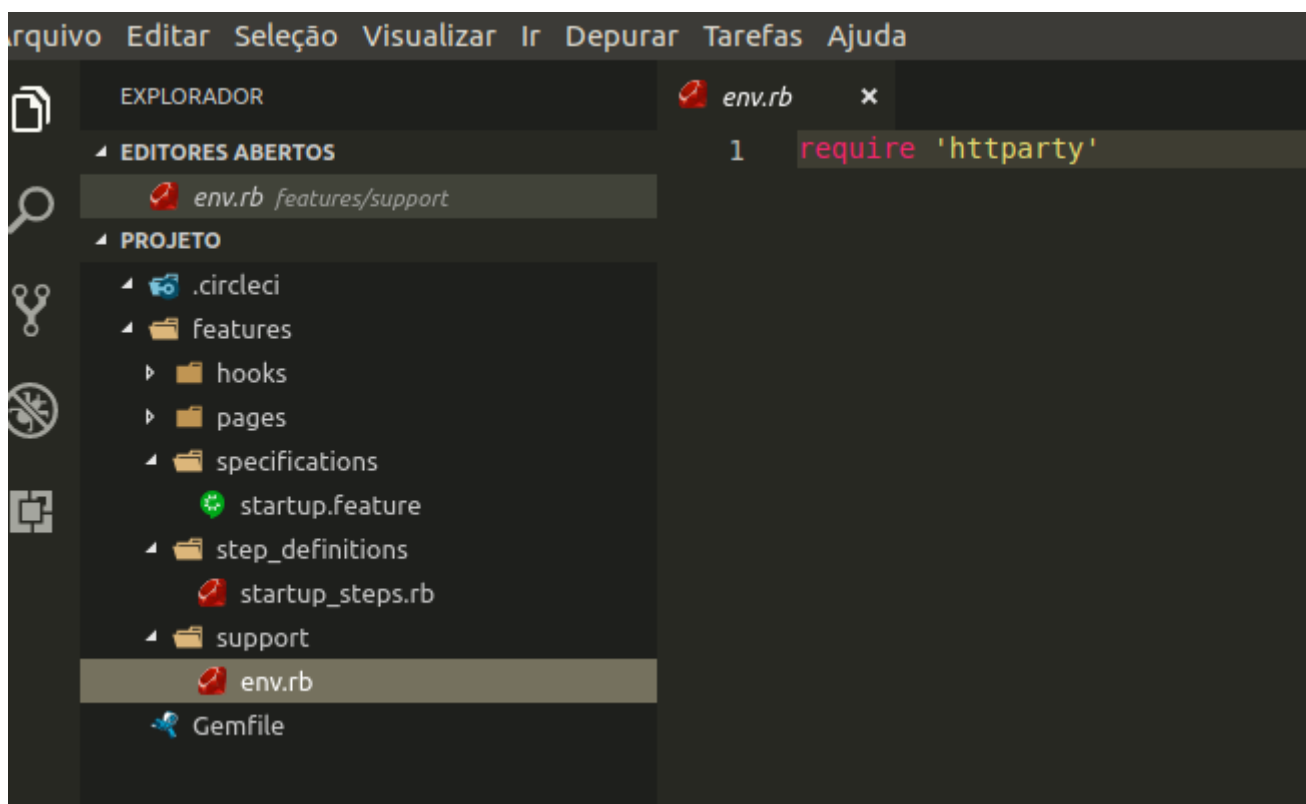
```
1 Dado("o endereço da API para manter o cadastro de StartupQ") do
2   pending # Write code here that turns the phrase above into concrete actions
3 end
4
5 Quando("realizar uma requisição para cadastrar uma startup") do
6   pending # Write code here that turns the phrase above into concrete actions
7 end
8
```

[Open in app](#)

Beleza, agora para nosso código rodar vamos colocar o código bem parecido ao nosso teste inicial dentro dos passos do nosso teste.

```
startup_steps.rb
1 Dado("o endereço da API para manter o cadastro de StartupQ") do
2   $uri_base = "http://5b49f9b0ff11b100149bf42b.mockapi.io/cm/startup"
3 end
4
5 Quando("realizar uma requisição para cadastrar uma startup") do
6   $response = HTTParty.post($uri_base, :body => {"nome":"Maximilhas","cidade":"Belo Horizonte"})
7 end
8
9 Então("a API irá retornar os dados do cadastro da Startup respondendo o código {int}") do |int|
10   puts "response body #{$response.body}"
11   puts "response code #{$response.code}"
12 end
```

Quase tudo pronto pra executar o teste, mas dentro da nossa estrutura de pastas existe um arquivo de configuração do ambiente que precisa saber que vamos usar o HTTParty ele fica em *features/support/env.rb*. Vamos editá-lo e colocar a linha de comando para importar a gem para nosso projeto.



require da gem do HTTParty

Agora sim tudo certo, vamos digitar 'cucumber' novamente pelo terminal e observar a execução do nosso primeiro teste.

[Open in app](#)


Funcionalidade: Manter dados de Startup através da API
 Como um usuário do sistema
 Eu quero realizar as requisições na API
 A fim de manipular as informações do cadastro de startup

```

  Cenário: Cadastrar uma Startup # features/specifications/startup.feature:8
  Dado o endereço da API para manter o cadastro de StartupQ # features/step definitions/startup_steps.rb:1
  Quando realizar uma requisição para cadastrar uma startup # features/step definitions/startup_steps.rb:5
  Então a API irá retornar os dados do cadastro da Startup respondendo o código 200 # features/step definitions/startup_steps.rb:9
    response body {"id":"3","data":"2018-07-15T20:46:45.573Z","nome":"Maximilhas","conta":"06197812","cidade":"Belo Horizonte"}
    response code 201

1 scenario (1 passed)
3 steps (3 passed)
0m1.397s
  
```

5b49f9b0ff11b100149bf42b.mockapi.io/cm/startup

```

[
  - {
    id: "1",
    data: "2018-07-14T15:22:16.680Z",
    nome: "conta.MOBI",
    conta: "27000720",
    cidade: "Nova Lima"
  },
  - {
    id: "2",
    data: "2018-07-15T09:19:24.577Z",
    nome: "symppla",
    conta: "21421999",
    cidade: "Belo Horizonte"
  },
  - {
    id: "3",
    data: "2018-07-15T20:46:45.573Z",
    nome: "Maximilhas",
    conta: "06197812",
    cidade: "Belo Horizonte"
  }
]
  
```

Muito bem! Ele cadastrou com sucesso em nosso primeiro teste, retornou os dados do cadastro com sucesso, mas sabemos que este teste, escrito dessa forma esta longe de ser a ideal, por isso vamos incrementar agora outros arquivos de configuração para nosso projeto.

Hooks

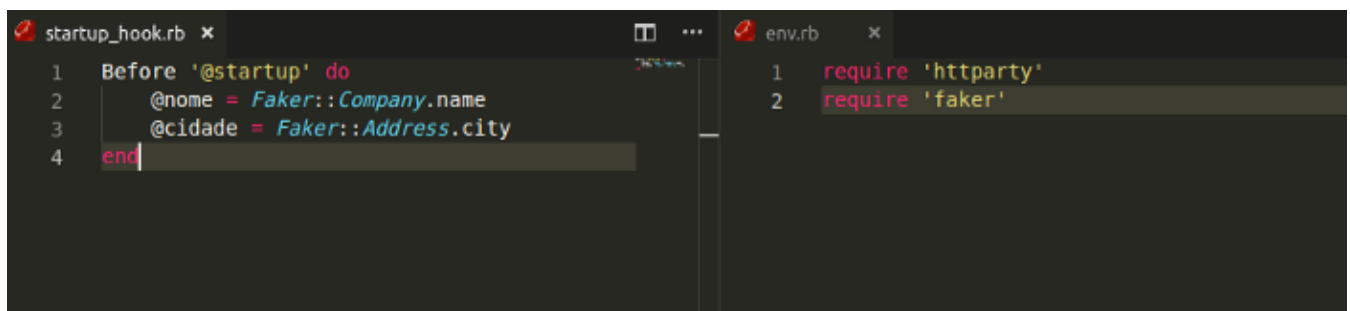
Vamos entender como instanciar variáveis para execução da nossa feature usando os hooks, e para isso vamos criar um novo arquivo dentro do diretório

[Open in app](#)

Antes de começarmos a escrever o código do nosso arquivo, vamos instalar mais uma gem bastante usada que se chama 'faker' usando o comando:

```
gem install faker
```

Esta gem é uma porta da biblioteca **Data::Faker** do *Perl* que gera dados falsos para preenchimento de campos para testes automatizados.

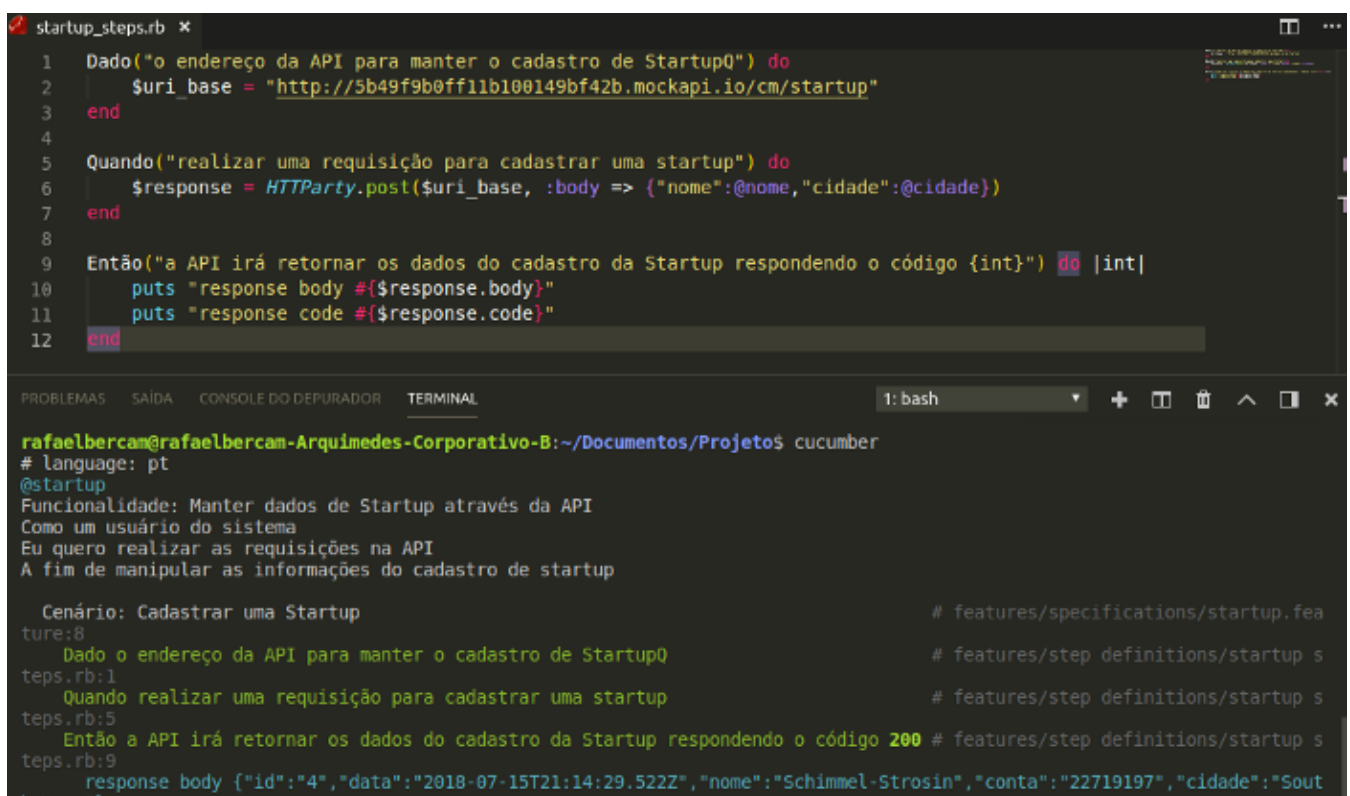


```
startup_hook.rb x
1 Before '@startup' do
2   @nome = Faker::Company.name
3   @cidade = Faker::Address.city
4 end

env.rb x
1 require 'httparty'
2 require 'faker'
```

Lembre-se de adicionar ao nosso arquivo *env.rb* a gem **faker** para ser reconhecida.

Então agora temos as variáveis '@nome' e a '@cidade' cadastrado com dados fakers para realizar os testes vamos alterar nosso arquivo '*features/step_definities/startup_steps.rb*' para receber essas variáveis.



```
startup_steps.rb x
1 Dado("o endereço da API para manter o cadastro de StartupQ") do
2   $uri_base = "http://5b49f9b0ff11b100149bf42b.mockapi.io/cm/startup"
3 end
4
5 Quando("realizar uma requisição para cadastrar uma startup") do
6   $response = HTTParty.post($uri_base, :body => {"nome":@nome,"cidade":@cidade})
7 end
8
9 Então("a API irá retornar os dados do cadastro da Startup respondendo o código {int}") do |int|
10  puts "response body #{$response.body}"
11  puts "response code #{$response.code}"
12 end

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL
1: bash
rafaelbercam@rafaelbercam-Arquimedes-Corporativo-B:~/Documentos/Projetos cucumber
# language: pt
@startup
Funcionalidade: Manter dados de Startup através da API
Como um usuário do sistema
Eu quero realizar as requisições na API
A fim de manipular as informações do cadastro de startup

Cenário: Cadastrar uma Startup # features/specifications/startup.feature:8
Dado o endereço da API para manter o cadastro de StartupQ # features/step definitions/startup_steps.rb:1
Quando realizar uma requisição para cadastrar uma startup # features/step definitions/startup_steps.rb:5
Então a API irá retornar os dados do cadastro da Startup respondendo o código 200 # features/step definitions/startup_steps.rb:9
response body {"id":"4","data":"2018-07-15T21:14:29.522Z","nome":"Schimmel-Strosin","conta":"22719197","cidade":"Souto Moura"}
```

[Open in app](#)

Requisição com dados Fake sendo realizada

Veja que bacana! Após alterar o corpo da requisição colocando as variáveis no lugar da string de nome e cidade que eram enviadas, quando rodar o Cucumber ele irá gerar dados aleatórios.

Ainda temos muito para melhorar neste teste, e usando da mesma forma que o padrão de **Page Objects** utiliza, vamos criar nossas pages para armazenar as requisições do HTTParty e para isso vamos criar um novo arquivo no diretório *'features/pages/startup_page.rb'*

Nesse arquivo vamos criar uma classe contendo os métodos da requisição que queremos, isso ajuda muito a deixar nosso código **dry** (Don't repeat yourself).

```
startup_page.rb x
1  class Startup
2    include HTTParty
3    require_relative '../hooks/startup_hook'
4    base_uri "http://5b49f9b0ff11b100149bf42b.mockapi.io/cm"
5
6    def initialize(body)
7      @options = {:body => body}
8    end
9
10   def postStartup
11     self.class.post("/startup", @options)
12   end
13 end
```

Nesta classe temos alguns elementos um pouco mais complexos, mas nada que complique nosso código. Para melhor aprendizado vale a consulta na documentação do [HTTParty](#).

- **base_uri** é uma classMethod do HTTParty que salva qual é a url padrão da nossa requisição dentro desta classe.
- o método **initialize** é um método construtor que quando instanciado preciso passar o corpo da requisição para ele.
- Já o método **postStartup** foi criado para realizar o método post passando apenas o **.endpoint** da nossa url e as opções carregadas na inicialização da nossa classe.

[Open in app](#)

```
startup_hook.rb x
1  Before '@startup' do
2
3      @nome = Faker::Company.name
4      @cidade = Faker::Address.city
5
6      @body = {
7          "nome": @nome,
8          "cidade": @cidade
9      }
10
11     @startup = Startup.new(@body)
12 end
```

Veja que agora nosso arquivo declarou um objeto '@body' e uma variável @startup instanciando um novo tipo **Startup** da nossa classe do **HTTParty** passando o @body como parâmetro.

Em alguns casos você irá precisar fazer uma conversão explícita para o formato **JSON** como no trecho abaixo:

```
startup_hook.rb ●
1  Before '@startup' do
2
3      @nome = Faker::Company.name
4      @cidade = Faker::Address.city
5
6      body = {
7          "nome": @nome,
8          "cidade": @cidade
9      }
10     @body = JSON.generate(body)
11
12     @startup = Startup.new(@body)
13
14 end
```

Por último vamos agora alterar nossos steps em `'features/step_definitions/startup_steps.rb'` para chamar nosso método.

[Open in app](#)

```
4
5 Quando("realizar uma requisição para cadastrar uma startup") do
6   $response = @startup.postStartup
7 end
8
9 Então("a API irá retornar os dados do cadastro da Startup respondendo o código {int}") do |int|
10   puts "response body #{ $response.body}"
11   puts "response code #{ $response.code}"
12 end
```

PROBLEMAS SAÍDA CONSOLE DO DEPURADOR TERMINAL 1: bash

```
# language: pt
@startup
Funcionalidade: Manter dados de Startup através da API
Como um usuário do sistema
Eu quero realizar as requisições na API
A fim de manipular as informações do cadastro de startup

  Cenário: Cadastrar uma Startup
  ture:8
    Dado o endereço da API para manter o cadastro de StartupQ
  teps.rb:1
    Quando realizar uma requisição para cadastrar uma startup
  teps.rb:5
    Então a API irá retornar os dados do cadastro da Startup respondendo o código 200 # features/step definitions/startup s
  teps.rb:9
    response body {"id":"5","data":"2018-07-16T09:51:30.161Z","nome":"nome 5","conta":"65166851","cidade":"cidade 5","{"
    nome\":"\\"Braun-Pagac\","cidade\":"\\"West Erlene\":"":""}
    response code 201

1 cenário (1 passed)
3 steps (3 passed)
0m2.007s
```

Veja que em nossos steps o endereço da api já foi abstraído pela nossa classe '**Startup**' em pages e não existe mais a necessidade de informar a url e para chamar o método, dessa forma, precisamos apenas chamar a variável **@startup** (instanciada no hook) e acessar o método **.postStartup** que criamos.

Ao rodar o cucumber o funcionamento do nosso teste passa normalmente!

Agora nosso projeto esta rodando com uma configuração bem melhor delegando responsabilidade para suas classes.

Specs

Para finalizar nosso teste a API precisa realizar alguns testes para garantir o que foi cadastrado, e para isso vamos usar o rspec do Ruby.

```
startup_hook.rb env.rb startup.feature startup_steps.rb x
```

```
4
5 Quando("realizar uma requisição para cadastrar uma startup") do
6   $response = @startup.postStartup
7 end
8
9 Então("a API irá retornar os dados do cadastro da Startup respondendo o código {int}") do |int|
10   #expect do status code e message
11   expect($response.code).to eq(201)
12   puts "Response code: #{ $response.code}"
13   expect($response.message).to eq("Created")
14   puts "Response Message: #{ $response.message}"
15
16   #imprime os atributos da requisição
17   puts "ID : #{ $response["id"]}"
18   puts "Data : #{ $response["data"]}"
```

[Open in app](#)


```

PROBLEMAS  SAÍDA  CONSOLA DO DEPURADOR  TERMINAL
1: bash
teps.rb:5
Então a API irá retornar os dados do cadastro da Startup respondendo o código 201 # features/step definitions/startup s
teps.rb:9
  Response code: 201
  Response Message: Created
  ID : 22
  Data : 2018-07-15T19:46:27.302Z
  Nome : Ortiz-Hahn
  Conta : 91980430
  Cidade: Lake Wes

1 scenario (1 passed)
3 steps (3 passed)
0m1.525s

```

Após implementar dois expects validando se o **status code** e **message** retornariam **201** e **Created** respectivamente, o teste foi executado e validado.

Fiz uma impressão acessando os atributos dos dados gerados pela API para mostrar que consigo abstrair estes dados e posso inclusive guarda-los em outras variáveis para um teste E2E.

HTTP Methods

Vamos validar os métodos disponíveis para realizar testes com o HTTParty

```

[
  Net::HTTP::Get,
  Net::HTTP::Post,
  Net::HTTP::Patch,
  Net::HTTP::Put,
  Net::HTTP::Delete,
  Net::HTTP::Head,
  Net::HTTP::Options,
  Net::HTTP::Move,
  Net::HTTP::Copy
]

```

Além do método get e post como vimos até aqui, o HTTParty da suporte a vários tipos de requisição e para isso vamos realizar testes nos seguintes .endpoints

- **GET** '/startup' => Retorna todos os registros
- **GET** '/startup/:id' => Retorna um registro de acordo com o ID
- **POST** '/startup' => Cadastra um novo registro
- **PUT** '/startup/:id' => Altera os dados do registro de acordo com o ID

[Open in app](#)

vamos criar novos cenários dentro do nosso arquivo `.feature`

```
startup.feature
1  #language: pt
2  @startup
3  Funcionalidade: Manter dados de Startup através da API
4  Como um usuário do sistema
5  Eu quero realizar as requisições na API
6  A fim de manipular as informações do cadastro de startup
7
8  Cenário: Cadastrar uma Startup
9      Dado o endereço da API para manter o cadastro de Startup
10     Quando realizar uma requisição para cadastrar uma startup
11     Então a API irá retornar os dados do cadastro da Startup respondendo o código 201
12
13  Cenário: Consultar uma Startup
14      Dado o endereço da API para manter o cadastro de Startup
15     Quando realizar uma requisição passando o ID da startup
16     Então a API irá retornar os dados da Startup correspondente respondendo o código 200
17
18  Cenário: Alterar uma Startup
19      Dado o endereço da API para manter o cadastro de Startup
20     Quando realizar uma requisição para alterar uma startup
21     Então a API irá retornar os dados da Startup alterados respondendo o código 200
22
23  Cenário: Deletar uma Startup
24      Dado o endereço da API para manter o cadastro de Startup
25     Quando realizar uma requisição para excluir uma startup
26     Então a API deverá retornar os dados da exclusão respondendo o código 200
27
```

E vamos criar também na nossa page de requisição novos métodos (`getStartup`, `putStartup` e `deleteStartup`) e estou passando o parâmetro `'id'` para ser acrescentado ao `.endpoint` das requisições.

```
startup_page.rb
1  class Startup
2      include HTTParty
3      require_relative '../hooks/startup_hook'
4      base_uri "http://5b49f9b0ff11b100149bf42b.mockapi.io/cm"
5
6      def initialize(body)
7          @options = {:body => body}
8          @options2 = {}
9      end
10
11     def postStartup
12         self.class.post("/startup", @options)
13     end
14
15     def getStartup (id)
16         self.class.get("/startup/#{id}", @options2)
17     end
18
19     def putStartup (id)
20         self.class.put("/startup/#{id}", @options)
21     end
22 end
```

[Open in app](#)


26

end

Repare que além de criar outros métodos eu criei uma **@options2** pois algumas requisições não há necessidade de passar o corpo como uma consulta por exemplo (GET).

Daí é só implementar os steps e pronto!

```
# language: pt
@startup
Funcionalidade: Manter dados de Startup através da API
Como um usuário do sistema
Eu quero realizar as requisições na API
A fim de manipular as informações do cadastro de startup

Cenário: Cadastrar uma Startup # features/specifications/startup.feature:8
  Dado o endereço da API para manter o cadastro de Startup # features/step_definitions/startup_steps.rb:1
  Quando realizar uma requisição para cadastrar uma startup # features/step_definitions/startup_steps.rb:5
  Então a API irá retornar os dados do cadastro da Startup respondendo o código 201 # features/step_definitions/startup_steps.rb:9
    Response code: 201
    Response Message: Created
    ID : 35
    Data : 2018-07-15T23:41:47.066Z
    Nome : Gulowski Group
    Conta : 13804109
    Cidade: Port Herrilee
```

```
Cenário: Consultar uma Startup # features/specifications/startup.feature:13
  Dado o endereço da API para manter o cadastro de Startup # features/step_definitions/startup_steps.rb:1
  Quando realizar uma requisição passando o ID da startup # features/step_definitions/startup_steps.rb:27
  Então a API irá retornar os dados da Startup correspondente respondendo o código 200 # features/step_definitions/startup_steps.rb:31
    ID : 35
    Data : 2018-07-15T23:41:47.066Z
    Nome : Gulowski Group
    Conta : 13804109
    Cidade: Port Herrilee
    Status Code: 200
```

```
Cenário: Alterar uma Startup # features/specifications/startup.feature:18
  Dado o endereço da API para manter o cadastro de Startup # features/step_definitions/startup_steps.rb:1
  Quando realizar uma requisição para alterar uma startup # features/step_definitions/startup_steps.rb:45
  Então a API irá retornar os dados da Startup alterados respondendo o código 200 # features/step_definitions/startup_steps.rb:49
    ID : 35
    Data : 2018-07-15T23:41:47.066Z
    Nome : Durgan-Lakin
    Conta : 13804109
    Cidade: Lailafurt
    Status Code: 200
```

```
Cenário: Deletar uma Startup # features/specifications/startup.feature:23
  Dado o endereço da API para manter o cadastro de Startup # features/step_definitions/startup_steps.rb:1
  Quando realizar uma requisição para excluir uma startup # features/step_definitions/startup_steps.rb:62
  Então a API deverá retornar os dados da exclusão respondendo o código 200 # features/step_definitions/startup_steps.rb:66
    ID : 35
    Data : 2018-07-15T23:41:47.066Z
    Nome : Durgan-Lakin
    Conta : 13804109
    Cidade: Lailafurt
    Status Code: 200

4 scenarios (4 passed)
12 steps (12 passed)
0m2.253s
```


[Open in app](#)

*Não se preocupe muito com o código, pois o mesmo estará disponível no meu repositório do git. Recomendo realizar o **clone** ou baixar o **zip** e descompactar no seu projeto para ver como tudo foi implementado.*

rbercam/httparty

httparty - Repositório da Postagem do Medium sobre HTTParty

github.com

CircleCI

Agora vamos falar sobre Entrega Contínua! O CircleCI testa automaticamente sua compilação em um contêiner limpo ou em uma máquina virtual. O que precisamos é criar o arquivo **config.yml** dentro do nosso repositório [**circleci**]

```
version: 2 # use CircleCI 2.0
jobs: # a collection of steps
  build:
    working_directory: ~/HTTParty # directory where steps will run
    docker: # run the steps with Docker
      - image: circleci/ruby:2.6-rc-node # ...with this image as the
primary container; this is where all `steps` will run
    environment: # environment variables for primary container
      BUNDLE_JOBS: 3
      BUNDLE_RETRY: 3
      BUNDLE_PATH: vendor/bundle
      PGHOST: 127.0.0.1
      PGUSER: circleci-demo-ruby
      RAILS_ENV: test
    steps: # a collection of executable commands
      - checkout # special step to check out source code to working
directory
      # Which version of bundler?
      - run:
        name: Which bundler?
        command: bundle -v

      # Restore bundle cache
      - restore_cache:
        keys:
          - rails-demo-bundle-v2-{{ checksum "Gemfile.lock" }}
          - rails-demo-bundle-v2-
```


[Open in app](#)


```
# Store bundle cache
- save_cache:
  key: rails-demo-bundle-v2-{{ checksum "Gemfile.lock" }}
  paths:
    - vendor/bundle

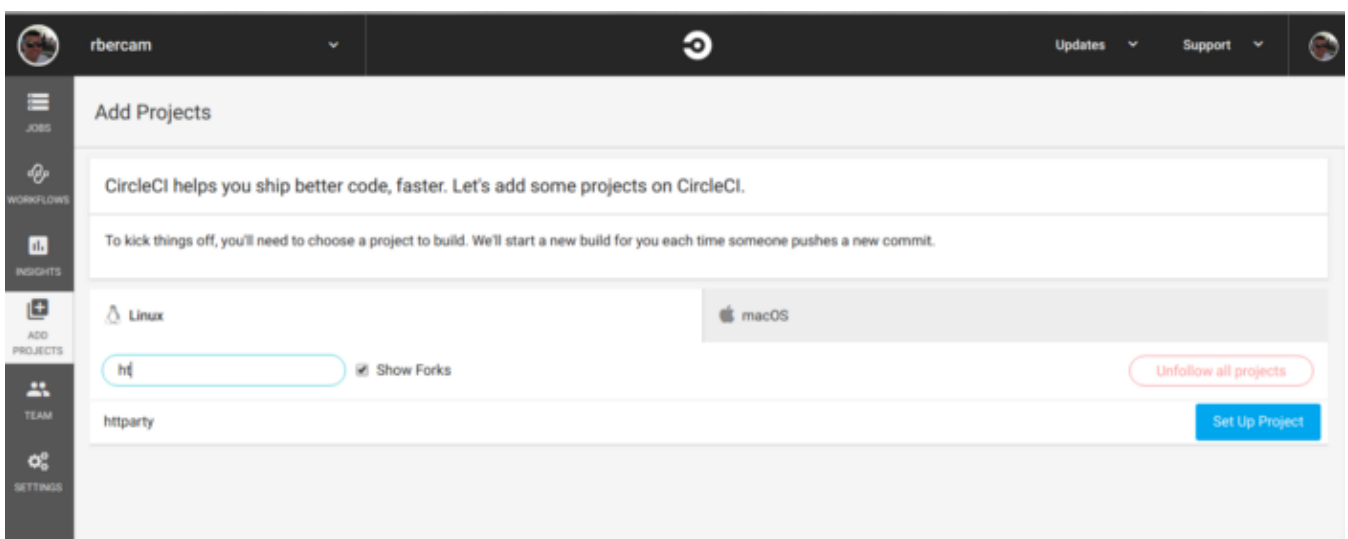
- run:
  name: Run Cucumber
  command: |
    bundle exec cucumber --format pretty --format html --
out=test_results/features_report.html

# Save test results for timing analysis
- store_test_results:
  path: ~/HTTParty/test_results
- store_artifacts:
  path: ~/HTTParty/test_results

workflows:
  version: 2

all_commits:
  jobs:
    - build:
      filters:
        branches:
          ignore:
            - developer
            - test

# See https://circleci.com/docs/2.0/deployment-integrations/
for example deploy configs
```



Após configurar o arquivo e commitar seu repositório no git vá até o dashboard do circleci e na opção **[ADD Projects]** filtre seu projeto e clique em **[Set Up Project]**

[Open in app](#)

You're almost there! We're going to walk you through setting up a configuration file, committing it, and turning on our listener so that CircleCI can test your commits.

Want to skip ahead? Jump right [into our documentation](#), set up a .yml file, and kick off your build with the button below.

⚠️ If you start building before you've added a config.yml file, this will start a project on CircleCI 1.0, which will no longer be supported after August 31, 2018. [Add a project on CircleCI 2.0.](#)

1. Create a folder named `.circleci` and add a file `config.yml` (so that the filepath be in `.circleci/config.yml`).
2. Populate the config.yml with the contents of the sample .yml (shown below). [Copy To Clipboard](#)
3. Update the sample .yml to reflect your project's configuration.
4. Push this change up to GitHub.
5. Start building! This will launch your project on CircleCI and make our webhooks listen for updates to your work. [Start building](#)

Sample .yml File

Na próxima tela clique em **[Start building]** para realizar a primeira build do projeto no ambiente de entrega contínua.

Workflows » [rbercam](#) » httparty

by project ▾ My branches All branches Showing 1-1

> httparty	<div><div><div><div><div></div><div>RUNNING</div><div>Cancel</div></div></div><div>master / all_commits</div></div></div>	<div><div>8 sec ago</div><div>00:07</div><div>f95193b</div></div>
------------	---	---

— Newer workflow runs Older workflow runs —

Veja que agora seu build esta sendo executado.

Jobs » [rbercam](#) » httparty » master » 5 (build) 2.0 [Rerun job with SSH](#) [n](#) [⚙️](#)

FIXED Finished: 6 min ago (02:01) Previous: 4 Parallelism: 1x out of 4x Queued: 00:00 waiting + 00:05 in queue Resources: 2CPU/4096MB Workflow: all_commits Context: N/A Triggered by: rbercam (pushed f2f064d)

COMMIT (1)
rafael.bercam f2f064d update

Test Summary Queue (00:05) Artifacts Configuration Timing Parameters

[Set Up Test Summary](#)

Show containers: All (1) Successful (1) Failed (0)

0 (02:01)	
TEST	
Spin up Environment	00:22
Checkout code	00:00
Which bundler?	00:00
Restoring Cache	00:00

[Open in app](#)

The screenshot shows the CircleCI web interface for a job named 'httparty' in the 'master' branch, build 5. The job status is 'passed' (green bar). The output shows two Cucumber scenarios: 'Alterar uma Startup' and 'Deletar uma Startup'. Both scenarios passed, with 12 steps each. The total execution time was 0m0.876s.

```
Jobs » rbercam » httparty » master » 5 (build) 2.0 Rerun job with SSH [n] [g]
```

```
0 (02:51) +
```

```
Conta : 01399577
Cidade: South Amybury
Status Code: 200

Cenário: Alterar uma Startup # features/specifications/startup.feature:18
  Dado o endereço da API para manter o cadastro de Startup # features/step_definitions/startup_steps.rb:1
  Quando realizar uma requisição para alterar uma startup # features/step_definitions/startup_steps.rb:45
  Então a API irá retornar os dados da Startup alterados respondendo o código 200 # features/step_definitions/startup_steps.rb:49
  ID : 35
  Data : 2018-07-16T09:06:03.889Z
  Nome : Green, Carter and Beahan
  Conta : 01399577
  Cidade: West Lawrence
  Status Code: 200

Cenário: Deletar uma Startup # features/specifications/startup.feature:23
  Dado o endereço da API para manter o cadastro de Startup # features/step_definitions/startup_steps.rb:1
  Quando realizar uma requisição para excluir uma startup # features/step_definitions/startup_steps.rb:62
  Então a API deverá retornar os dados da exclusão respondendo o código 200 # features/step_definitions/startup_steps.rb:66
  ID : 35
  Data : 2018-07-16T09:06:03.889Z
  Nome : Green, Carter and Beahan
  Conta : 01399577
  Cidade: West Lawrence
  Status Code: 200

4 scenarios (4 passed)
12 steps (12 passed)
0m0.876s
```

Ao final seu projeto estará sendo executado e monitorado através do CircleCI a cada commit que for realizado.

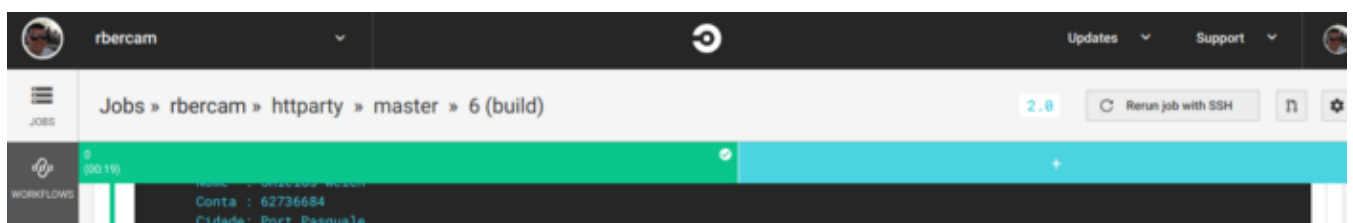
Configurando relatório Cucumber

Vamos adicionar no arquivo o comando para gerar o relatório e depois subir esse artefato como evidência do nosso teste, para isso basta colocar a linha de comando no nosso arquivo config.yml.

```
bundle exec cucumber --format pretty --format html --
out=test_results/features_report.html
```

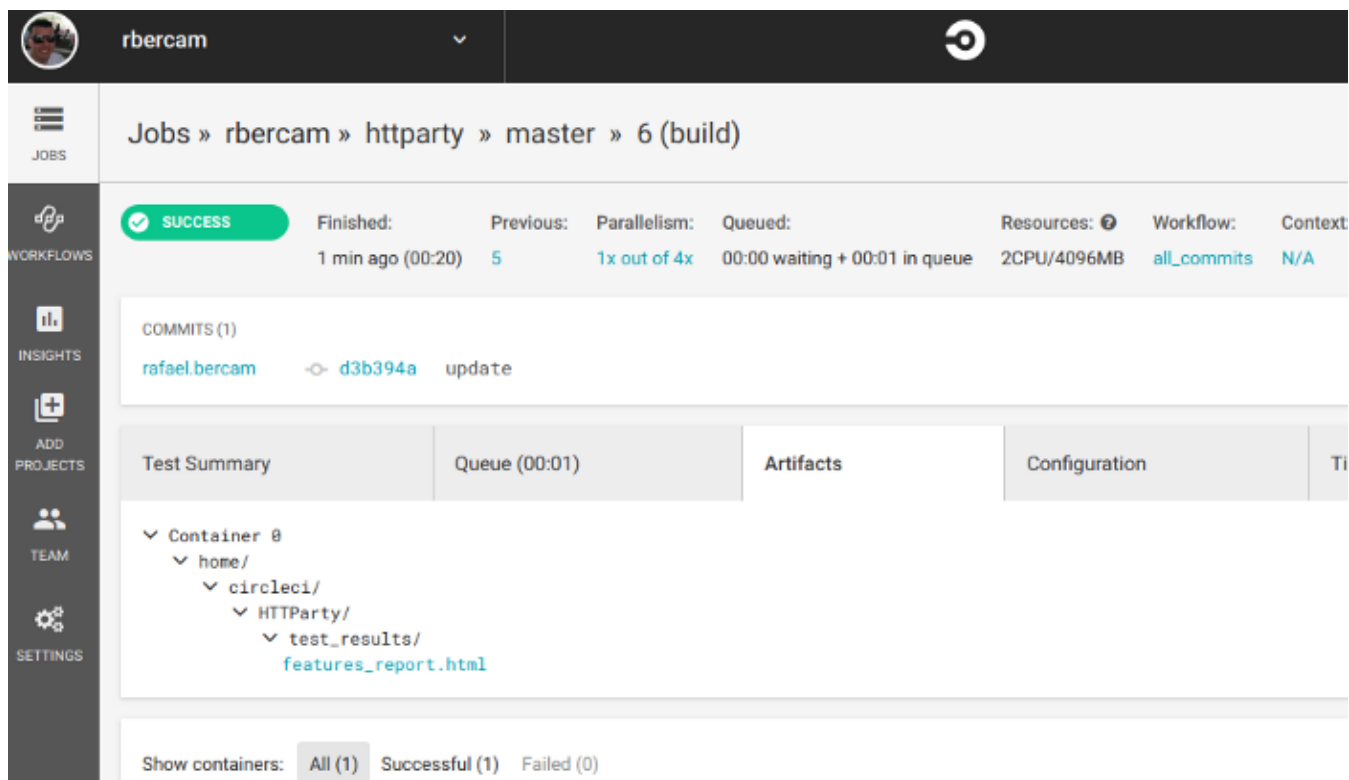
Para realizar um teste podemos executar este comando também no nosso projeto e ver a exibição deste relatório.

Após commitar as alterações vamos verificar se o artefato foi colhido de acordo com nossa configuração.



[Open in app](#)


Após a execução o circleci executou o uploading dos artefatos de teste então na aba Artifacts podemos constatar o upload do nosso relatório



Relatório

