

How to monitor a Rails API — [Part I]



Ashish Rao

Sep 19 · 10 min read

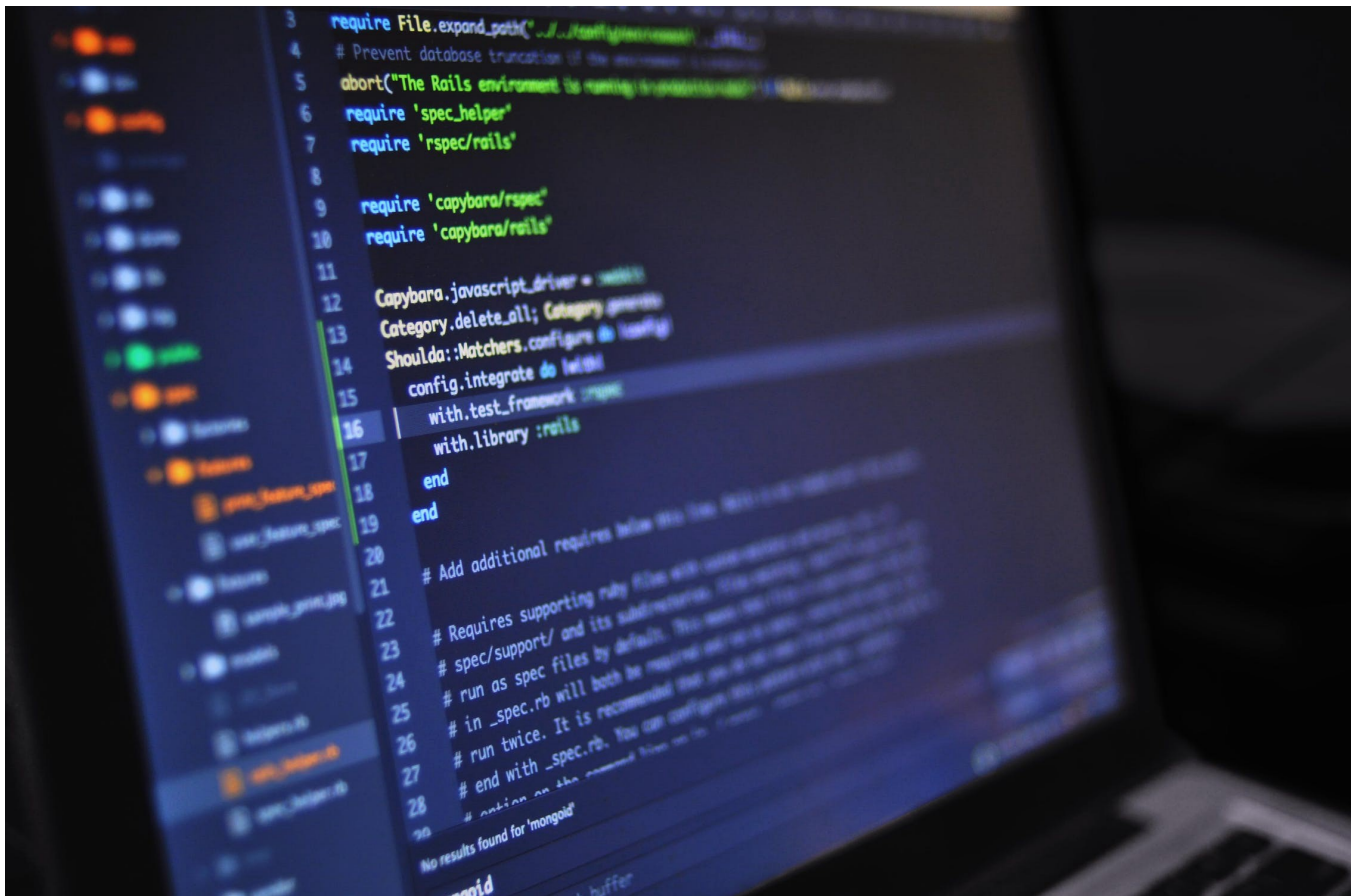


Photo by Luis gomes on Pexels

Rails is a popular web application framework. Chances are, on your software development journey you may have heard about Rails and may have built a traditional server-rendered web application.

As of version 5, Rails core now supports API only applications, previously we had to rely on an external gem `rails-api` which has since been merged to core rails.

In this two part blog post, we'll build a todo list API and figure out how to monitor the application using `StatsD`. I am not going to dive deep into this topic by including

authentication or pagination, The aim in this post is to just build a working API. Sound good? Okay here we go!

API endpoints

Search this file...		
1	Endpoint	Functionality
2	GET /todos	List all todos
3	POST /todos	Create a new todo
4	GET /todos/:id	Get a todo
5	PUT /todos/:id	Update a todo
6	DELETE /todos/:id	Delete a todo and its items
7	GET /todos/:id/items	Get a todo item
8	PUT /todos/:id/items	Update a todo item
9	DELETE /todos/:id/items	Delete a todo item

todo_endpoints.csv hosted with ❤️ by GitHub [view raw](#)

API Endpoints

Setup

Make sure Rails is installed with the latest version along with the latest version of Ruby.

```
$ rails -v
# => Rails 6.0.0

$ ruby -v
# => ruby 2.6.3p62
```

Generate a new project `todos-api` by running:

```
rails new todo-api --api --tests
```

`--api` argument tells Rails that we want an API application.

`--tests` argument tells Rails to exclude the default Minitest framework because we will use RSpec to test our API

Dependencies

- `rspec-rails` — Testing framework.
- `factory_bot_rails` — A fixtures replacement with a more straightforward syntax. You'll see.
- `shoulda-matchers` — Provides RSpec with additional matchers.
- `database_cleaner` — You guessed it! It literally cleans our test database to ensure a clean state in each test suite.
- `faker` — A library for generating fake data. We'll use this to generate test data.

Let's set them up. In your `Gemfile`:

Add `rspec-rails` to both the `:development` and `:test` groups.

```
# Gemfile
group :development, :test do
  gem 'rspec-rails', '~> 3.5'
end
```

Add `factory_bot_rails`, `shoulda-matchers`, `faker` and `database_cleaner` to the `:test` group.

```
# Gemfile
group :test do
  gem 'factory_bot_rails', '~> 4.0'
  gem 'shoulda-matchers', '~> 3.1'
  gem 'faker'
  gem 'database_cleaner'
end
```

Do a `$ bundle install`

Initialize the `spec` directory (where our tests will reside).

```
$ rails generate rspec:install
```

This adds the following files which are used for configuration:

- .rspec
- spec/spec_helper.rb
- spec/rails_helper.rb

Create a `factories` directory (factory bot uses this as the default directory). This is where we'll define the model factories.

```
$ mkdir spec/factories
```

Configuration

In `spec/rails_helper.rb`

```
# require database cleaner at the top level
require 'database_cleaner'

# configure shoulda matchers to use rspec as the test framework and
# full matcher libraries for rails
Shoulda::Matchers.configure do |config|
  config.integrate do |with|
    with.test_framework :rspec
    with.library :rails
  end
end

RSpec.configure do |config|
  # add `FactoryBot` methods
  config.include FactoryBot::Syntax::Methods

  config.before(:suite) do
    DatabaseCleaner.clean_with(:truncation)
    DatabaseCleaner.strategy = :transaction
  end

  config.around(:each) do |example|
    DatabaseCleaner.cleaning do
      example.run
    end
  end
end
```

Here we are just configuring Factory Bot and Shoulda Matchers methods to work with our specs.

Models

Let's start by generating the `Todo` model

```
$ rails g model Todo title:string created_by:string
```

And now the `Item` model

```
$ rails g model Item name:string done:boolean todo:references
```

By adding `todo:references` we are telling the generator to set up an association with the `Todo` model. This will do the following:

- Add a foreign key column `todo_id` to the `items` table
- Setup a `belongs_to` association in the `Item` model.

Finally, Let's run the migrations

```
$ rails db:migrate
```

We believe in TDD

Let's write the model specs first

1. `spec/models/todo_spec.rb`

```
require 'rails_helper'

RSpec.describe Todo, type: :model do

  it { should have_many(:items).dependant(:destroy) }

  it { should validate_presence_of(:title) }

  it { should validate_presence_of(:created_by) }

end
```

2. spec/models/item_spec.rb

```
require 'rails_helper'

RSpec.describe Item, type: :model do

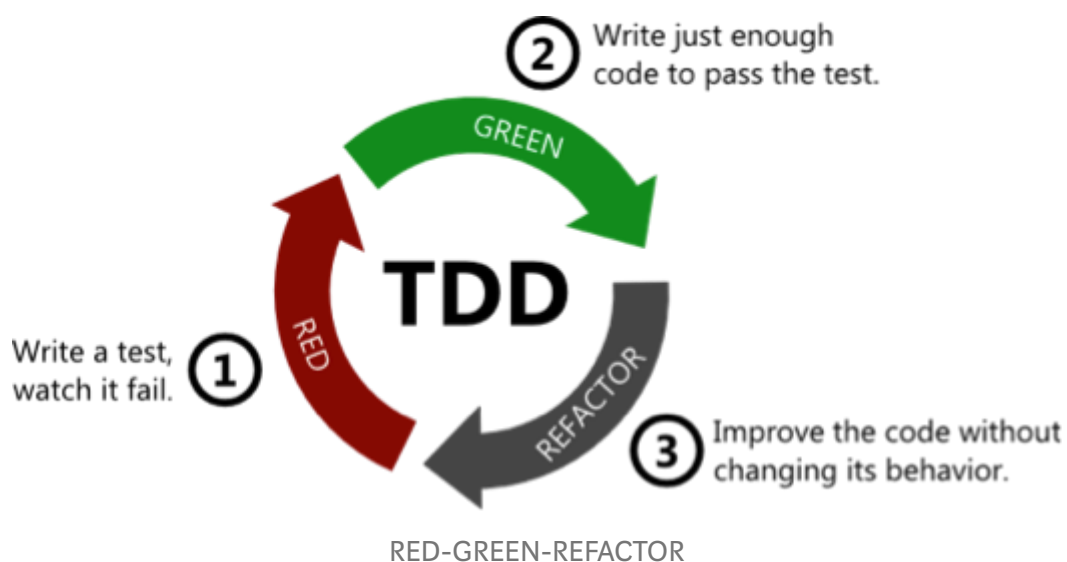
  it { should belong_to(:todo) }

  it { should validate_presence_of(:name) }

end
```

The shoulda-matchers gem provides RSpec with the nifty association and validation matchers above.

If we execute the specs we will obviously run into failures, but that's the protocol we have to follow.



Let's go ahead and fix the failures

```
# app/models/todo.rb

class Todo < ApplicationRecord
  has_many :items, dependent: :destroy

  validates_presence_of :title, :created_by
end

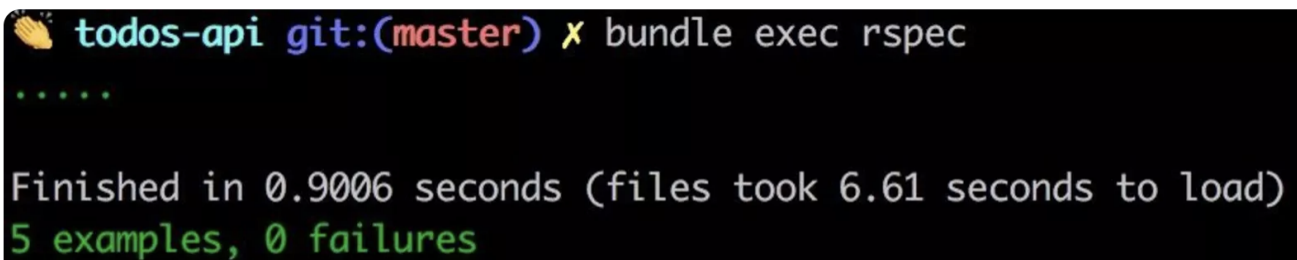
---
```

```
# app/models/item.rb
class Item < ApplicationRecord
  belongs_to :todo

  validates_presence_of :name
end
```

Now if we run our tests

```
$ bundle exec rspec
```



```
👉 todos-api git:(master) ✗ bundle exec rspec
.....

Finished in 0.9006 seconds (files took 6.61 seconds to load)
5 examples, 0 failures
```

And we pass the test 🤖

Controllers

So far we have set up our models and and successfully passed the tests for it as as well. Now let's set up our controllers.

```
$ rails g controller Todos
$ rails g controller Items
```

As usual we will be writing tests first, but we won't be writing controller specs instead we will write request specs instead.

Request specs can hit the applications' HTTP endpoints as opposed to controller specs which call methods directly. Since we are building an API application, this is exactly the kind of behaviour we want from our tests.

Add a `requests` folder to the `spec` directory with the corresponding spec files.

```
$ mkdir spec/requests && touch  
spec/requests/{todos_spec.rb,items_spec.rb}
```

Before we define the request specs, Let's add the model factories which will provide the test data.

Add the factory files:

```
$ touch spec/factories/{todos.rb,items.rb}
```

Define the factories

```
# spec/factories/todos.rb  
FactoryBot.define do  
  factory :todo do  
    title { Faker::Lorem.word }  
    created_by { Faker::Number.number(digits: 20) }  
  end  
end  
  
end  
end
```

We are ensuring that faker generates dynamic data every time the factory is invoked by wrapping it in a block. This way, we always have a unique data.

```
# spec/factories/items.rb  
FactoryBot.define do  
  factory :item do  
    name { Faker::Movies::StarWars.character }  
    done { false }  
    todo_id { nil }  
  end  
end
```

Todo API

Now we will be writing request specs for CRUD functionality

open up spec/requests/todos_spec.rb


```
require 'rails_helper'

RSpec.describe 'Todos API', type: :request do
  # initialize test data
  let!(:todos) { create_list(:todo, 20) }
  let(:todo_id) { todos.first.id }

  # Test suite for GET /todos
  # Test suite for GET /todos/:id
  # Test suite for POST /todos
  # Test suite for PUT /todos/:id
  # Test suite for DELETE /todos/:id

end
```

Above we are describing the type of test, in our case it is request spec

Test suite for GET /todos and GET /todos/:id

```
describe 'GET /todos' do
  # make HTTP get request before each example
  before { get '/todos' }

  it 'returns todos' do
    # Note `json` is a custom helper to parse JSON responses
    expect(json).not_to be_empty
    expect(json.size).to eq(20)
  end

  it 'returns status code 200' do
    expect(response).to have_http_status(200)
  end
end

describe 'GET /todos/:id' do
  before { get "/todos/#{todo_id}" }

  context 'when the record exists' do
    it 'returns the todo' do
      expect(json).not_to be_empty
      expect(json['id']).to eq(todo_id)
    end

    it 'returns status code 200' do
      expect(response).to have_http_status(200)
    end
  end

  context 'when the record does not exist' do
    let(:todo_id) { 100 }

    it 'returns status code 404' do
```

```
      expect(response).to have_http_status(404)
    end

    it 'returns a not found message' do
      expect(response.body).to match(/Couldn't find Todo/)
    end
  end
end
```

When making a GET request to /todos endpoint, as per our initialized data it should display 10 records and should return the status code 200.

When making a GET request to /todos/:id endpoint, if the record exists at the initialized id i.e id = 1, the response body should not be empty and return status code of 200.

Test suite for POST /todos

```
describe 'POST /todos' do
  # valid payload
  let(:valid_attributes) { { title: 'Learn Node', created_by: '1' } }

  context 'when the request is valid' do
    before { post '/todos', params: valid_attributes }

    it 'creates a todo' do
      expect(json['title']).to eq('Learn Node')
    end

    it 'returns status code 201' do
      expect(response).to have_http_status(201)
    end
  end

  context 'when the request is invalid' do
    before { post '/todos', params: { title: 'Apple' } }

    it 'returns status code 422' do
      expect(response).to have_http_status(422)
    end

    it 'returns a validation failure message' do
      expect(response.body)
        .to match(/Validation failed: Created by can't be blank/)
    end
  end
end
```

We have initialised a data object with title and created_by. We then make a POST request and pass the data object as params. The consecutive `it` block tests the endpoint by checking if the todo was created and if not then check for appropriate http status codes i.e 201 for created, 422 for unprocessable entity.

Test suite for PUT /todos/:id

```
describe 'PUT /todos/:id' do
  let(:valid_attributes) { { title: 'Shopping' } }

  context 'when the record exists' do
    before { put "/todos/#{todo_id}", params: valid_attributes }

    it 'updates the record' do
      expect(response.body).to be_empty
    end

    it 'returns status code 204' do
      expect(response).to have_http_status(204)
    end
  end
end
```

We are testing if we can update an existing record by making a PUT request. a data object with title as ‘Shopping’ is passed as params while making the request.

Test suite for DELETE /todos/:id

```
describe 'DELETE /todos/:id' do
  before { delete "/todos/#{todo_id}" }

  it 'returns status code 204' do
    expect(response).to have_http_status(204)
  end
end
```

Testing if a delete request is able to delete a particular record.

In a nutshell, We start by populating the database with a list of 10 todo records. We also have a custom helper method `json` which parses the JSON response to a Ruby Hash which is easier to work with in our tests. We will define it in `spec/support/request_spec_helper`

Finally,

```
$ bundle exec rspec
```

Our tests are gonna fail. That's because we haven't defined the routes yet.

Lets define them in `config/routes.rb`

```
# config/routes.rb
Rails.application.routes.draw do
  resources :todos do
    resources :items
  end
end
```

In our route definition, we're creating todo resource with a nested items resource. This enforces the one to many associations at the routing level.

Now, let's define the controller methods to finally get a green light in our tests

```
# app/controllers/todos_controller.rb
class TodosController < ApplicationController
  before_action :set_todo, only: [:show, :update, :destroy]

  # GET /todos
  def index
    @todos = Todo.all
    json_response(@todos)
  end

  # POST /todos
  def create
    @todo = Todo.create!(todo_params)
    json_response(@todo, :created)
  end

  # GET /todos/:id
  def show
    json_response(@todo)
  end

  # PUT /todos/:id
  def update
    @todo.update(todo_params)
    head :no_content
  end
end
```

```
# DELETE /todos/:id
def destroy
  @todo.destroy
  head :no_content
end

private

def todo_params
  # whitelist params
  params.permit(:title, :created_by)
end

def set_todo
  @todo = Todo.find(params[:id])
end
end
```

If you notice we have a new helper `json_response` which responds with JSON and HTTP status code. We will define this method in concerns folder.

```
# app/controllers/concerns/response.rb
module Response
  def json_response(object, status = :ok)
    render json: object, status: status
  end
end
```

Note: ActiveRecord will throw an exception `ActiveRecord::RecordNotFound` in cases where the record does not exist when we try to find a todo by id.

Handling record not found errors

```
# app/controllers/concerns/exception_handler.rb
module ExceptionHandler
  # provides the more graceful `included` method
  extend ActiveSupport::Concern

  included do
    rescue_from ActiveRecord::RecordNotFound do |e|
      json_response({ message: e.message }, :not_found)
    end

    rescue_from ActiveRecord::RecordInvalid do |e|
      json_response({ message: e.message }, :unprocessable_entity)
    end
  end
end
```

Note: In the create method in TodosController, we are using create! which will raise an exception ActiveRecord::RecordInvalid. Hence we rescue from this exception as well.

Our controller classes aren't aware of such helpers yet. Let's fix it by including these modules in the application controller.

```
# app/controllers/application_controller.rb
class ApplicationController < ActionController::API
  include Response
  include ErrorHandler
end
```

If we go and run `$ bundle exec rspec`

```
.....
Finished in 0.35909 seconds (files took 1.32 seconds to load)
18 examples, 0 failures
```

And we pass! 🙌

TodosItems API

```
# spec/requests/items_spec.rb
require 'rails_helper'

RSpec.describe 'Items API' do
  # Initialize the test data
  let!(:todo) { create(:todo) }
  let!(:items) { create_list(:item, 20, todo_id: todo.id) }
  let(:todo_id) { todo.id }
  let(:id) { items.first.id }

  # Test suite for GET /todos/:todo_id/items
  describe 'GET /todos/:todo_id/items' do
    before { get "/todos/#{todo_id}/items" }

    context 'when todo exists' do
      it 'returns status code 200' do
        expect(response).to have_http_status(200)
      end

      it 'returns all todo items' do
```

```
      expect(json.size).to eq(20)
    end
  end

  context 'when todo does not exist' do
    let(:todo_id) { 0 }

    it 'returns status code 404' do
      expect(response).to have_http_status(404)
    end

    it 'returns a not found message' do
      expect(response.body).to match(/Couldn't find Todo/)
    end
  end
end

# Test suite for GET /todos/:todo_id/items/:id
describe 'GET /todos/:todo_id/items/:id' do
  before { get "/todos/#{todo_id}/items/#{id}" }

  context 'when todo item exists' do
    it 'returns status code 200' do
      expect(response).to have_http_status(200)
    end

    it 'returns the item' do
      expect(json['id']).to eq(id)
    end
  end

  context 'when todo item does not exist' do
    let(:id) { 0 }

    it 'returns status code 404' do
      expect(response).to have_http_status(404)
    end

    it 'returns a not found message' do
      expect(response.body).to match(/Couldn't find Item/)
    end
  end
end

# Test suite for PUT /todos/:todo_id/items
describe 'POST /todos/:todo_id/items' do
  let(:valid_attributes) { { name: 'Visit Narnia', done: false } }

  context 'when request attributes are valid' do
    before { post "/todos/#{todo_id}/items", params:
valid_attributes }

    it 'returns status code 201' do
      expect(response).to have_http_status(201)
    end
  end

  context 'when an invalid request' do
```

```

before { post "/todos/#{todo_id}/items", params: {} }

it 'returns status code 422' do
  expect(response).to have_http_status(422)
end

it 'returns a failure message' do
  expect(response.body).to match(/Validation failed: Name
can't be blank/)
end
end
end

# Test suite for PUT /todos/:todo_id/items/:id
describe 'PUT /todos/:todo_id/items/:id' do
  let(:valid_attributes) { { name: 'Mozart' } }

  before { put "/todos/#{todo_id}/items/#{id}", params:
valid_attributes }

  context 'when item exists' do
    it 'returns status code 204' do
      expect(response).to have_http_status(204)
    end

    it 'updates the item' do
      updated_item = Item.find(id)
      expect(updated_item.name).to match(/Mozart/)
    end
  end

  context 'when the item does not exist' do
    let(:id) { 0 }

    it 'returns status code 404' do
      expect(response).to have_http_status(404)
    end

    it 'returns a not found message' do
      expect(response.body).to match(/Couldn't find Item/)
    end
  end
end

# Test suite for DELETE /todos/:id
describe 'DELETE /todos/:id' do
  before { delete "/todos/#{todo_id}/items/#{id}" }

  it 'returns status code 204' do
    expect(response).to have_http_status(204)
  end
end
end

```

The tests at this point would fail, lets define the todo items controller.


```
# app/controllers/items_controller.rb
class ItemsController < ApplicationController
  before_action :set_todo
  before_action :set_todo_item, only: [:show, :update, :destroy]

  # GET /todos/:todo_id/items
  def index
    json_response(@todo.items)
  end

  # GET /todos/:todo_id/items/:id
  def show
    json_response(@item)
  end

  # POST /todos/:todo_id/items
  def create
    @todo.items.create!(item_params)
    json_response(@todo, :created)
  end

  # PUT /todos/:todo_id/items/:id
  def update
    @item.update(item_params)
    head :no_content
  end

  # DELETE /todos/:todo_id/items/:id
  def destroy
    @item.destroy
    head :no_content
  end

  private

  def item_params
    params.permit(:name, :done)
  end

  def set_todo
    @todo = Todo.find(params[:todo_id])
  end

  def set_todo_item
    @item = @todo.items.find_by!(id: params[:id]) if @todo
  end
end
```

Moment of truth...

```
.....
Finished in 0.78446 seconds (files took 1.47 seconds to load)
```

34 examples, 0 failures

And here we go, we have with us a working todo API.

That's it for part one. At this point we have covered

1. Generate an API application with Rails 6
2. Setup RSpec with Factory Bot, Database Cleaner, Shoulda Matchers and Faker.
3. Build models and controllers with TDD

In the next part, we shall see how we can setup up some monitoring for our API application from scratch. Hope to see you there. Cheers! 🍺

Like this article? Follow me on twitter.

Link to the Github repo.

References

Here are some useful links which I came across when I started with building APIs :-

<https://medium.com/pixelpoint/oh-man-look-at-your-api-22f330ab80d5>

<https://scotch.io/tutorials/build-a-restful-json-api-with-rails-5-part-one>

<https://medium.com/@kiddy.xyz/tutorial-restful-api-dengan-ruby-on-rails-4-mysql-part-2-create-update-delete-c0db70d17a84>

<https://medium.com/@lukepierotti/setting-up-rspec-and-factory-bot-3bb2153fb909>

[Ruby on Rails](#) [Statsd](#) [Grafana](#) [Influxdb](#) [Telegraf](#)

[About](#) [Help](#) [Legal](#)