Mosaico Weather API: desenvolvimento de web API's em R para acesso à dados de previsão do tempo

Igor Scheuermann

Curso de Ciência da Computação – Universidade de Passo Fundo (UPF)

Campus 1 - BR 285 - Passo Fundo (RS) - Brasil

147183@upf.br

Abstract: *This pager introduce an approach to develop APIs using the R programming language through the Plumber package. This package has tools for develop functions, that can be executed on a server and answer WEB requests. To demonstrate the capabilities of Plumber, some tests were made using datasets of Brazilian weather forecast, generated and provided by CPTEC/INPE. The tests made possible access that data through an R API, generating output in tables, graphics and maps. Finally, with Plumber, developers can use this strategy to universalize their data to be used.*

Resumo: *Este trabalho apresenta uma abordagem para o desenvolvimento de RESTful API´s utilizando a linguagem R por meio do pacote plumber. Este pacote possui ferramentas para geração de funções responsivas a requisições WEB e permite que sua execução seja realizada em um servidor. Com o intuito de demonstrar as capacidades deste pacote, alguns testes foram realizados sob um conjunto de dados que contém a previsão do tempo do Brasil gerada pelo CPTEC/INPE. Os testes possibilitaram o acesso a estes dados via uma API´s R, gerando como saída tabelas, gráficos e mapas. Por fim, com o uso do plumber, desenvolvedores em R poderão utilizar-se dessa estratégia para universalizar a forma de consumo dos dados gerados por suas funções.*

1. Introdução

As APIs (*Application Programming Interface*) são uma forma de integrar diferentes sistemas e aplicações. Elas proporcionam que sistemas que operem em linguagens diferentes possam trocar informações de forma eficiente e segura. Assim, desenvolvedores têm a possibilidade de utilizar sistemas heterogêneos para desenvolvimento, utilizando a linguagem mais apropriada para determinada tarefa. As APIs ficam transparentes ao usuário, permitindo uma interação mais natural.

Uma API pode ser uma ferramenta de uso local ou online. No primeiro caso, acontece na máquina do usuário, que irá interligar sistemas internos, sejam do SO ou de seus softwares, assim como, por exemplo, o relógio do sistema, que pode ser associado a outros softwares executados na máquina. Porém, ela vem sendo aplicada imensamente em ferramentas providas pela internet. Segundo a *ProgrammableWeb*[1] [Wendell 2017],

já existem mais de 17 mil API disponíveis para uso na WEB em 2017, e estes estão tendo uma crescente exponencial ao longo dos anos. Uma API WEB é acessada por meio de requisições, que contém 4 requisitos: URL (endereço acessado); Método (tipo de requisição); *Header* (cabeçalho de informações do cliente); e *Body* (parâmetros da requisição).

Para este trabalho foram desenvolvidas soluções em ambiente de codificação R. Com o uso do pacote Plumber, foram disponibilizados dados de previsão do tempo, dispostos em arquivos binários do R na forma de uma API, de maneira a permitir toda a versatilidade e o desempenho desta linguagem para ferramentas que não são naturalmente compatíveis, por meio de requisições WEB, demonstrando a validade desta forma de implementação frente a outras alternativas.

Assim, o trabalho será descrito em cinco seções, sendo que na seção 2 serão detalhados os componentes presentes em uma requisição HTTP e as possíveis vantagens na vinculação de APIs com a linguagem R. Na seção 3 apresenta-se o pacote Plumber, suas funcionalidades, modo de utilização e a sua respectiva disponibilização em um servidor. Na seção 4 são apresentados alguns exemplos de utilização do serviço com os arquivos binários em R contendo dados de previsão do tempo pertencentes a base de dados do Mosaico (Grupo de Pesquisa em Modelagem e Simulação em Ambiente Interativo e Computacional) da Universidade de Passo Fundo. Na última seção serão apresentadas as conclusões do trabalho.

2. Acesso a dados via API no R

Com a disseminação da internet, pode-se universalizar funções e recursos já existentes, sem que tudo precise ser desenvolvido novamente em outras ferramentas. As APIs Web servem para, de forma padronizada, disponibilizar recursos existentes de um serviço para outros por meio de requisições. Isso permite que sistemas escritos em qualquer linguagem possam obter dados, sem se importarem com a linguagem de origem.

Requisições HTTP funcionam da seguinte maneira [Zavadniak 2017]: Um cliente faz uma solicitação a um servidor, que irá interpretar essa solicitação e fornecer uma resposta adequada. Quando uma URL é acessada em algum navegador, por exemplo, é enviada uma requisição ao servidor e a resposta será recebida e interpretada por este navegador. A requisição possui várias informações, como a URL, que define o endereço da requisição; o método (GET, POST, DELETE, PUT), o header, contendo as informações do cliente solicitante; e o body, onde são passados possíveis parâmetros e seus valores. A resposta contém um cabeçalho de informações e o corpo, com os dados propriamente ditos.

A linguagem de programação R [R Core Team 2008] possui vários recursos que facilitam a manipulação, análise e visualização de dados. Isso faz com que se veja interessante a possibilidade de disponibilizar esses dados via Web, de forma que possam ser compartilhados, como, por exemplo, um conjunto de dados de previsão do tempo que irá disponibilizar esses[1] dados ao público em geral. Ferramentas comuns para exibição de dados de previsão do tempo são *dashboards* com mapas interativos, como o Shiny [Winston Chang et al. 2019]. No entanto, os dados disponibilizados por estas

[1] Disponível em https://www.programmableweb.com/api-research

ferramentas, por mais que disponham de certa interatividade, impedem com que esses dados sejam consumidos em sua forma mais bruta, por algo ou alguém que queira manipular diretamente e fazer suas próprias rotinas analíticas.

Por isso, uma alternativa é a integração da manipulação de dados pela linguagem R com a utilização de APIs via WEB. Assim, usado como meio de campo entre a manipulação dos dados e a forma de disponibilizá-los, a API serve para unificar a forma de disponibilização para que qualquer linguagem possa acessar e manipular, inclusive o próprio R.

3. O pacote Plumber

Pacotes, em R, são códigos já programados que servem para serem incorporados e solucionar situações de codificação. O pacote Plumber [Trestle Technology 2017] nasceu na iniciativa de disponibilizar dados trabalhados em R via Web e, assim como é padrão de APIs Web, ele trabalha com todos os tipos de requisição. O plumber executa em uma porta que pode ser definida manualmente. Caso não informada, a porta selecionada será aleatória, entre as disponíveis no sistema. Ele roda um código programado em R, onde funções irão tratar as requisições que forem solicitadas.

O Plumber permite transformar códigos R para responder a requisições WEB meramente adicionando anotações como cabeçalho das funções. Este cabeçalho transforma a função no chamado *endpoint*, e determina a sua URL. Além disso, também define o(s) tipo(s) de requisição que a função suporta, sendo elas GET, POST, PUT, DELETE ou HEAD. Na Figura 1, é exibido o exemplo de um endpoint, capaz de aceitar requisições de três tipos diferentes, cuja URL se dá pelo caminho `/plot`.

```
19
20  #' @get /plot
21  #' @post /plot
22  #' @put /plot
23  function(){
24    ...
25    ...
26  }
27
```

Figura 1: Exemplo de um endpoint com três tipos de requisição

Um endpoint poderá ter ainda a sua rota definida de forma dinâmica. Isso permite com que os programadores tenham maior flexibilidade em definir o caminho que o endpoint deverá acolher. Assim, definindo o setor do endereço entre os sinais de menor e maior, o texto do endereço é interpretado pelo servidor e poderá servir como variável dentro do código do programa, permitindo um reaproveitamento de código, definindo variações de forma dinâmica. No exemplo da Figura 2, é possível visualizar uma rota dinâmica onde a variável *rda* irá receber o valor informado na requisição e seguir o fluxo conforme o seu valor. Por padrão, o dado recebido será considerado uma string. Porém, outro tipo de dado (inteiro; booleano; numérico) poderá ser definido manualmente, assim a requisição será interpretada única e somente se o conteúdo desta seja compatível.

```
2
3  load("./data/prec.Rda")
4  load("./data/ur2m.Rda")
5  load("./data/ocis.Rda")
6
7
8
9  #' @get /plot/<rda>
10 function(rda){
11   selec <- switch(rda,
12                   prec = prec,
13                   ur2m = ur2m,
14                   ocis = ocis)
15   return(selec)
16 }
17
```

Figura 2: Exemplo de rota dinâmica na URL da função *plot*

Outra funcionalidade deste pacote são os filtros [Trestle Technology 2017], procedimentos que irão lidar com todas as requisições enviadas ao sistema. Isso permite que os desenvolvedores criem passos a serem seguidos a cada requisição ao servidor. Esses filtros possuem três motivos centrais para sua utilização. Primeiro, a possibilidade de manipular e colher algum *log*, e liberar a requisição para o próximo passo; Segundo, responder a requisição por conta própria e encerrar a sua interpretação; E terceiro, gerar um erro.

O primeiro dos casos é o mais comum. Captar e modificar a requisição ou então como um *logger,* como demonstrado na Figura 3. Com a utilização de variável com valor por parâmetro, qualquer alteração no conteúdo destes serão visíveis para qualquer outro filtro ou endpoint. Além disso, esses filtros são comumente usados para utilizar informações já acumuladas em cookies e disponibilizar esses valores aos endpoints.

O segundo e terceiro caso são mais raros, mas ainda assim relevantes. Um filtro pode responder sozinho a requisição, em casos onde, por exemplo, se necessite de autenticação para a execução ser prosseguida. Assim, caso isto não seja validado, o filtro envia essa resposta e encerra o fluxo do código, ou até mesmo lançar um erro.

```
29
30 #' @filter logger
31 function(req){
32   cat(as.character(Sys.time()), "-",
33       req$REQUEST_METHOD, req$PATH_INFO, "-",
34       req$HTTP_USER_AGENT, "@", req$REMOTE_ADDR, "\n")
35   plumber::forward()
36 }
37
```

Figura 3: Exemplo de filtro de log

O Plumber também é capaz de interpretar *query strings* [Trestle Technology 2017] inseridos na requisição HTML. Isso possibilita que o serviço que está

requisitando possa inserir dados extras, não obrigatórios na rota original codificada. Assim quando essa informação é inserida, o Plumber irá interpretar para o servidor e atribuir para a variável correspondente o valor informado. Para tal, essas variáveis precisam estar descritas como parâmetros da função, com um valor já pré definido, como demonstrado na Figura 4. Assim, qualquer definição de variável na requisição que seja inexistente no código será simplesmente ignorada.

```
40
41  #' @get /
42  search <- function(X=0, Y=0){
43    paste0("O valor de X é '", X, "'. ",
44           "O valor de Y é '", Y, "'.")
45  }
46
47  ##  Acessando http://localhost:8000/?X=7&Y=32 irá mostrar:
48  ##  ["O valor de X é '7'. O valor de Y é '32'."]
49
```

Figura 4: Exemplo de *Query String*, com as variáveis *X* e *Y*

Por mais útil que possam ser as *query strings*, existem situações em que a quantidade de informações a serem transpassadas na requisição são em demasia, o que, além não ser prático se inserirem todos os dados diretamente no navegador WEB, podem ocorrer problemas de limitações acerca do tamanho da string em certos navegadores. Assim o Plumber também possui suporte a *Request Body* [Trestle Technology 2017], que são mais comumente utilizadas via requisições do tipo POST, mas não exclusivamente. Para tal, são utilizados outros artifícios para envio, como por exemplo o comando *curl*[2].

Após receber a requisição e ter toda a cadeia de tratamento dentro do código R, é necessário gerar uma resposta em um formato que seja familiar ao cliente, que irá recebê-las. Desta forma, o JSON (JavaScript Object Notation) torna-se uma boa alternativa para formatar a saída de dados, haja vista que este é o formato padrão adotado por grande parte das APIs Web. Portanto, por padrão, o Plumber irá serializar as saídas de dados nesse formato, utilizando o pacote R *jsonlite*. Não obstante, certas formas de resposta não se dão viáveis via JSON. Pode-se necessitar gerar uma imagem que foi renderizada no R, ou então a emissão de uma página HTML. Desta forma, o Plumber disponibiliza uma série de outras formas de renderização, tais quais html, jpeg, png e html widget.

É também possível passar a resposta da requisição sem serialização. Assim, basta retornar diretamente o objeto no final do endpoint, que este será enviado ao cliente sem qualquer cabeçalho ou serialização. Além disso, Plumber também permite utilizar uma anotação para definir manualmente o cabeçalho, definindo o tipo de arquivo a ser retornado. Assim, por exemplo, pode ser gerado um pdf em R e, quando esta função for requisitada, o cliente irá receber o arquivo no formato pdf, sem serialização.

Para que o arquivo de código R seja compilado com os atributos e funções do plumber, ele é invocado pela função `plub("nome_do_arquivo.R")`. Neste

[2] Disponível em https://curl.haxx.se/

momento, os trechos de códigos globais serão interpretados imediatamente, e na sua execução serão executados e as variáveis instanciadas. Já as funções características do Plumber, tais quais os endpoints, somente serão interpretadas quando estas forem invocadas por uma requisição.

Como a linguagem R é single-thread, mesmo com a utilização do Plumber, esta poderá somente executar uma tarefa por vez. Assim, o indicado é evitar que as funções da API possuam processamentos demorados de serem executados, deixando essa parte já executada no corpo da carga desta.

Após a criação de toda a API, é necessário executá-la em algum servidor, para que esta possa ser acessada por outros usuários. Ela pode ser executada numa máquina local, simplesmente usando o comando `run()`, abrindo assim uma conexão na porta 8000. Porém, APIs web são desenvolvidas comumente para utilização na WEB, sendo assim necessário a configuração de um servidor à parte. O Plumber pode ser hospedado por algum servidor utilizando várias ferramentas. Uma das mais simples delas consiste na utilização do RStudio Connect [RStudio Team 2015], uma das plataformas do RStudio que suporta vários conteúdos R, incluíndo Plumber. Ele irá gerenciar automaticamente o fluxo de conexão, a escalabilidade dos processos para balancear o tráfego na rede e o controle de acessos.

Para hospedar a API diretamente no servidor, sem utilização de softwares autorais e dedicados, é necessário o uso de outras ferramentas. Como exemplo pode-se citar o *pm2*[3], que inicialmente foi desenvolvido para gerenciar processos em Node.js, mas pode ser adaptado para suportar serviços em R. O pm2 é um pacote mantido no *npm*[4], portanto o servidor precisará ter *Node.js* instalado. Ele permitirá que o serviço seja executado assim que o sistema inicializar. Como o p2m não é capaz de interpretar scripts R de forma nativa, é necessário definir um intérprete personalizado. Neste caso, faz-se necessário a criação de um script R que irá requisitar o pacote plumber e rodar o arquivo com o código, assim como mostrado na Figura 5. Após, basta executar o comando de execução da seguinte forma: `pm2 start --interpreter="Rscript" script.R`.

```
1
2  library(plumber)
3  r <- plumb("weather.R")
4  r$run(port=4000)
5
```

Figura 5: Script para execução da API em Plumber

4. Estudo de caso

Com o objetivo de verificar as funcionalidades do pacote plumber, foi desenvolvido uma API em R, com o objetivo de disponibilizar dados de previsões de tempo gerados pelo modelo Eta 15km do CPTEC (Centro de Previsão de Tempo e Estudos Climáticos) do INPE (Instituto Nacional de Pesquisas Espaciais) [Mesinger et al. 2016; Mesinger et al. 2012; Chou et al. 2005]. Estes dados de previsão do tempo, que estão armazenados

[3] Disponível em https://pm2.io/
[4] Disponível em https://www.npmjs.com/

diretamente na base de dados do servidor do CPTEC/INPE, foram baixados para o servidor do Mosaico e convertidos em arquivos no formato binário do R (.Rda). Com isso, vislumbrou-se a possibilidade da utilização desta forma de disponibilização destes tipos de dados frente a outras alternativas.

Para este estudo foram desenvolvidas cinco funções para demonstração da funcionalidade do pacote Plumber:

a) /plot/<rda>/<datahora>/<estado>: retorna um mapa de uma região/variável;

b) /dataframe/<lat>/<lon>/<var>/<itime>/<ftime>: retorna os dados de uma coordenada;

c) /plotvar/<lat>/<lon>/<rda>/<itime>/<ftime>: retorna dados de uma variável;

d) /plotthp/<lat>/<lon>/<itime>/<ftime>: retorna dados de três variáveis;

e) /meteogram/<lat>/<lon>/<itime>/<ftime>: retorna dados de cinco variáveis;

Para a execução destas funções são necessários os pacotes rgdal [Bivand; Keitt; Rowlingson 2019], sp [Bivand; Pebesma; Rubio 2013], raster [Hijmans 2019] e ggplot2 [Wickham 2016]. As funções são descritas a seguir:

a) /plot/<var>/<datahora>/<estado>: Função responsável por retornar a imagem de um mapa de uma determinada região do Brasil, com a informação de uma determinada variável climática em uma hora específica. Para a geração do mapa dos estados, foi utilizado um shapefile do Brasil. Os parâmetros são todos obrigatórios, sendo eles <estado>, região a ser exibida; <datahora>, dia e hora de referência no formato "AAAAMMDDHH"; e <var>, variável climática a ser exibida, admitindo esta os seguintes valores:
   - V10M - vento a 10m de altura em m/s;
   - TP2M - temperatura a 2m de altura;
   - PREC - precipitação;
   - UR2M - umidade a 2 m de altura;
   - OCIS - radiação solar.

   No exemplo da Figura 6, é possível visualizar a requisição no topo da imagem, onde o primeiro parâmetro escolhido é a variável, TP2M; o segundo a hora, 00 hrs do dia 14/05/2019; e o terceiro a região, RS (Rio Grande do Sul). Assim, é retornado o mapa com as informações requisitadas.

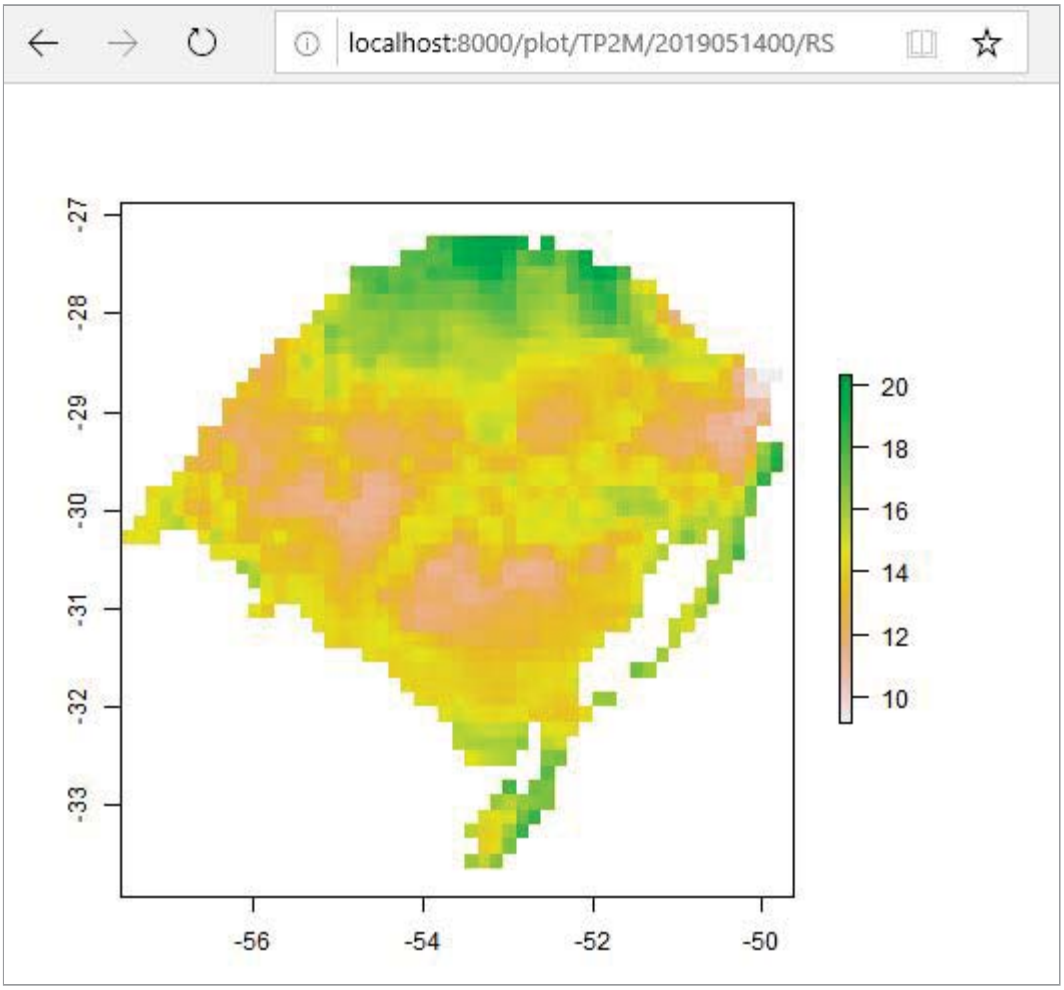

Figura 6: Chamada e retorno da função *plot*

b) /dataframe/<lat>/<lon>/<var>/<itime>/<ftime> : Função responsável por retornar um dataframe em formarto JSON, de acordo com a latitude, longitude, variável e intervalo de data e hora desejados. A latitude e longitude informadas são adaptadas para o ponto válido mais próximo, já que o modelo em grade possui resolução de 15 km x 15 km e o dado disponibilizado é o centro desta grade. Assim como na função anterior, são aceitos os mesmos valores na variável climática.

No exemplo da Figura 7, a requisição informada no topo gera o retorno, que contém os dados da latitude e longitude mais próximas as informadas na requisição da variável TP2M no intervalo de dias informados.

Figura 7: Chamada e retorno da função *dataframe*

c) /plotvar/<lat>/<lon>/<rda>/<itime>/<ftime>: Função responsável por retornar um gráfico contendo os valores da variável climática requisitada, de acordo com a latitude, longitude e intervalo de data e hora desejados. As regras de latitude,

longitude e variável climática são as mesmas da função anterior.

No exemplo da Figura 8, o retorno contém o gráfico do valor da temperatura a 2 metros de altura na latitude e longitude mais próximas as informadas na requisição no intervalo de 12 a 15 de maio de 2019.

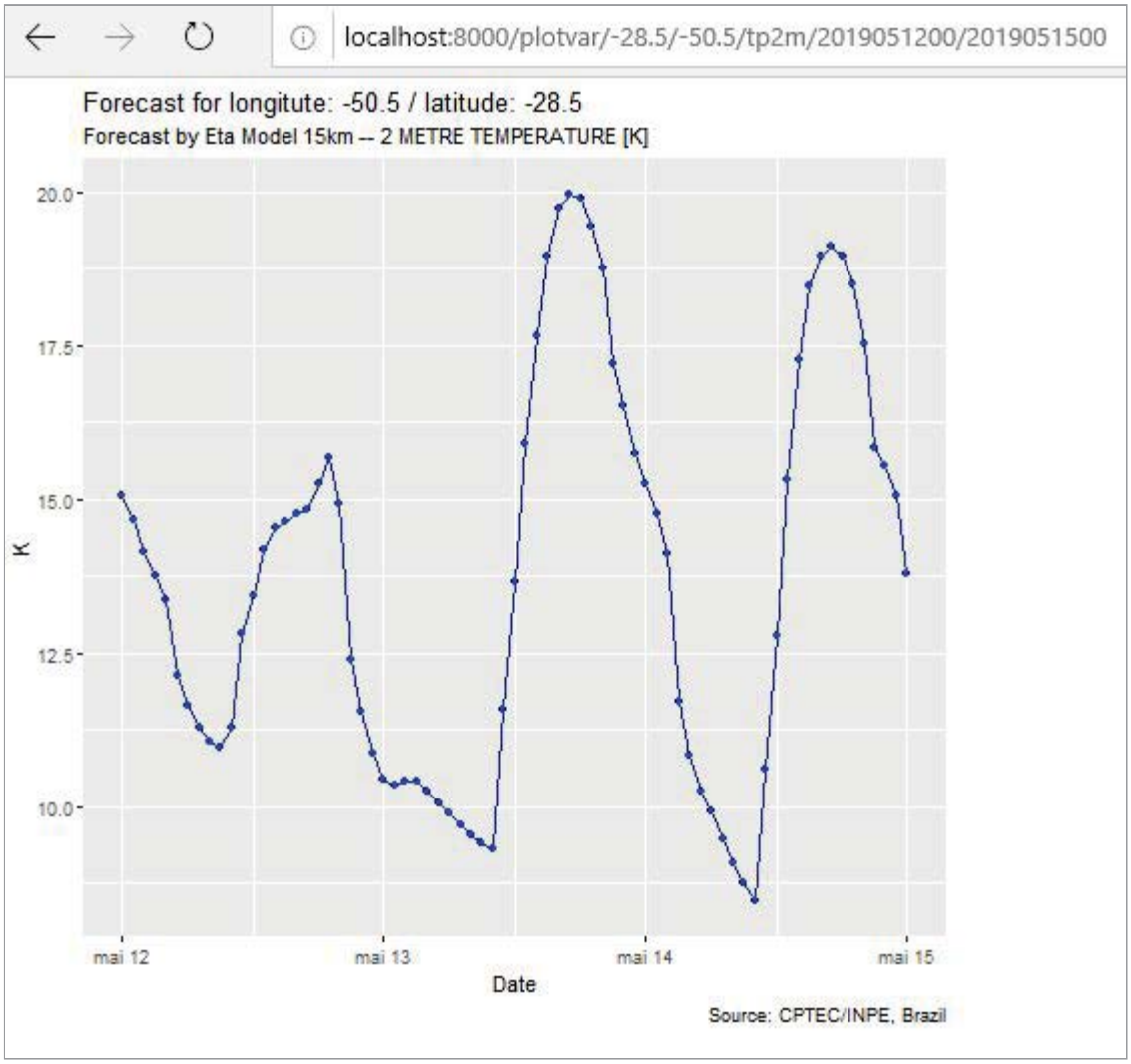

Figura 8: Chamada e retorno da função *plotvar*

d) /plotthp/<lat>/<lon>/<itime>/<ftime>: Função responsável por retornar um gráfico contendo os valores de três variáveis climáticas, sendo elas PREC, TP2M e UR2M, de acordo com a latitude, longitude e intervalo de data e hora desejados.

No exemplo da Figura 9, o retorno contém o gráfico do valor das 3 variáveis na latitude e longitude em questão no intervalo de 12 a 15 de maio de 2019, conforme informado na requisição.

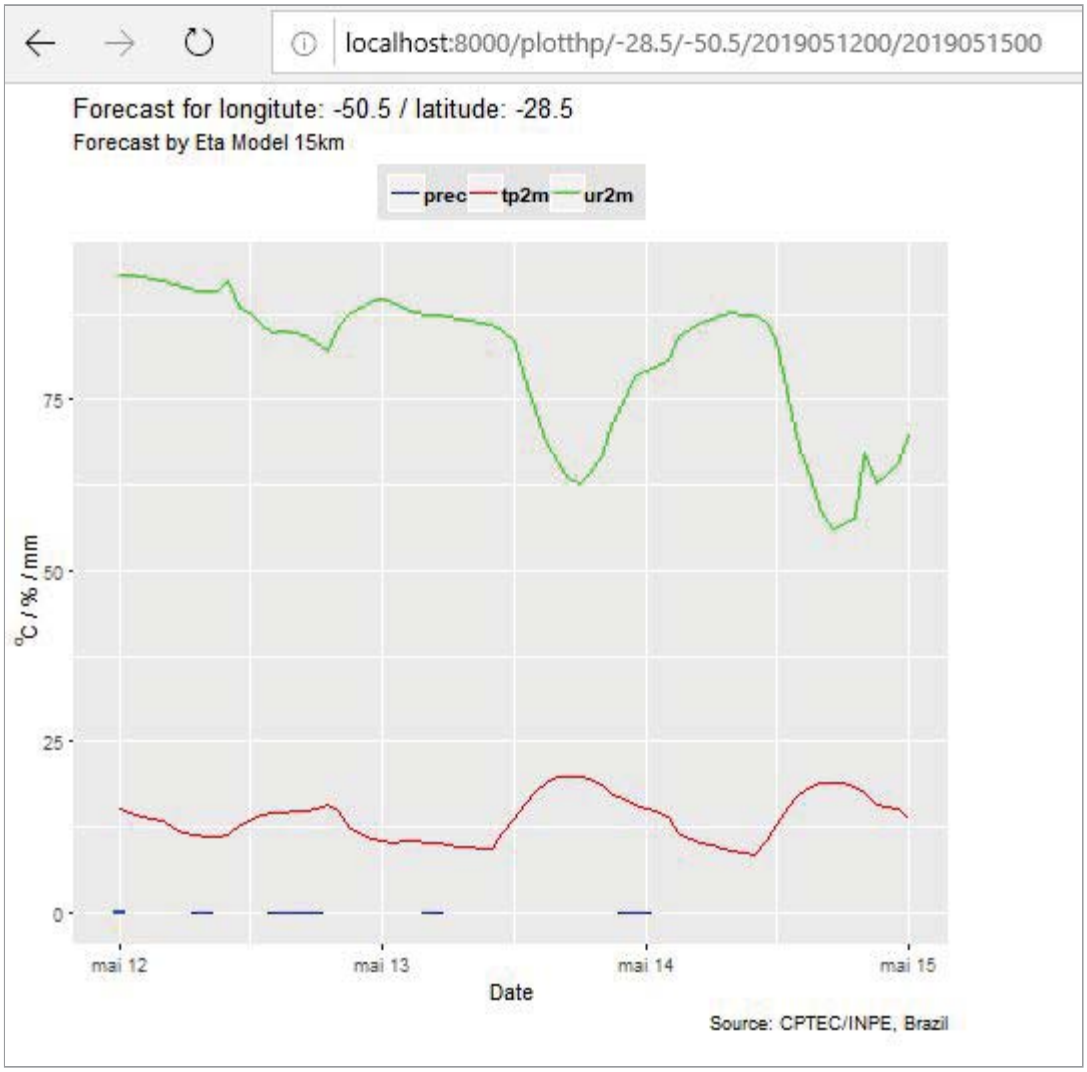

Figura 9: Chamada e retorno da função *plotthp*

e) /meteogram/<lat>/<lon>/<itime>/<ftime>: Função responsável por gerar um meteograma contendo os valores das cinco variáveis climáticas disponíveis, também de acordo com a latitude, longitude e intervalo de data e hora desejados.

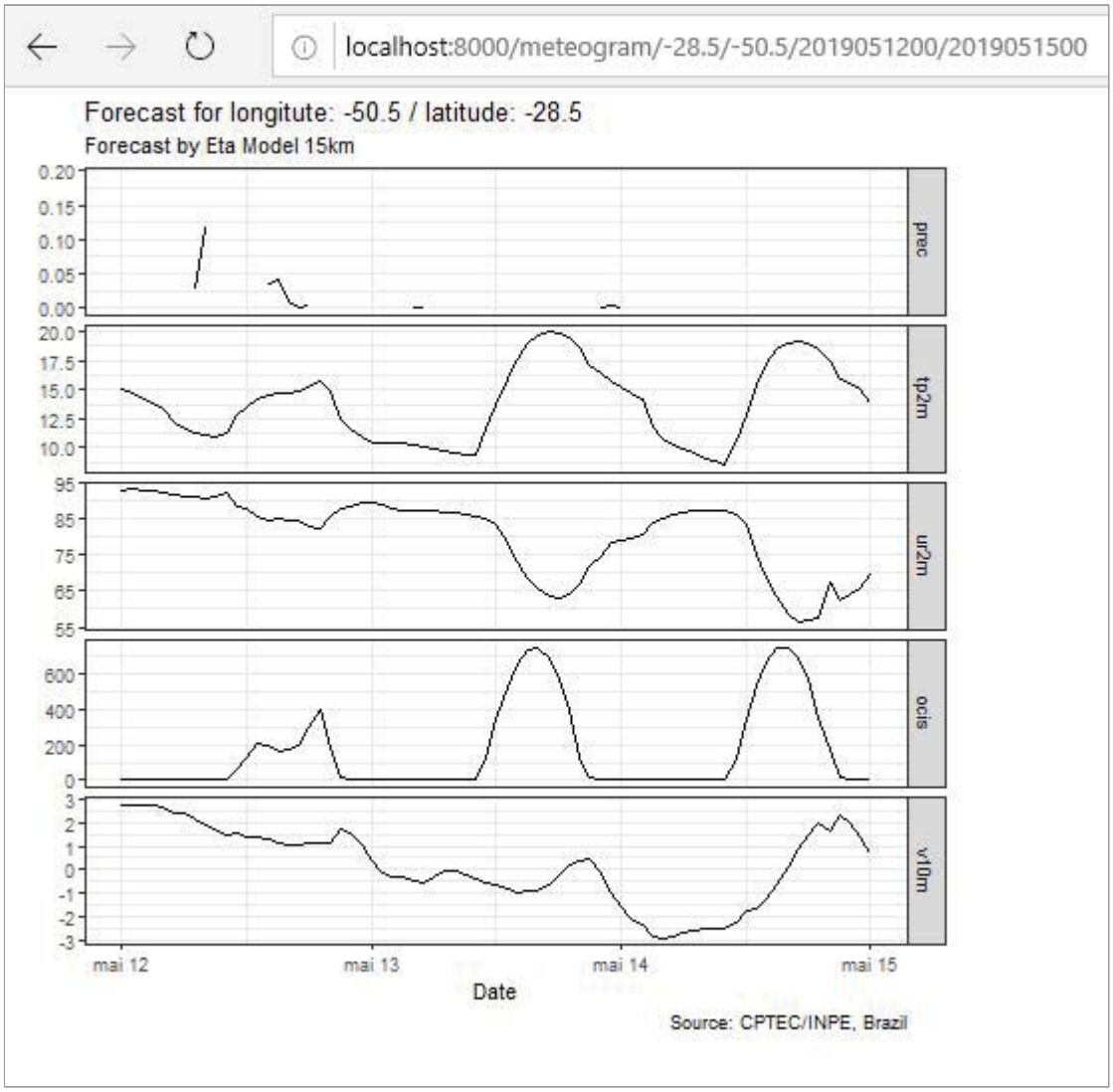

Figura 10: Chamada e retorno da função meteogram

No exemplo da Figura 10, o retorno exibe o meteograma, exibindo o valor das 5 variáveis na latitude/longitude em questão, no intervalo de 12 a 15 de maio de 2019, conforme requisição.

Em todos os exemplos demonstrados, as informações em questão já estavam todas carregadas em memória, no momento em que o serviço é executado no servidor ou na máquina local. Isso fez com que o carregamento e a exibição do resultado tenha o custo de tempo apenas relativo a renderização do retorno (algo que se mostra muito eficiente em R) e a conexão entre cliente/servidor.

5. Conclusões e trabalhos futuros

O pacote Plumber mostra uma grande gama de funcionalidades para tornar programas desenvolvidos em linguagem R disponíveis na forma de uma API, com várias possibilidades de tornar esse serviço disponível pela web. Assim, a linguagem R pode ser utilizada para acessar uma grande quantidade de dados, com as facilidades que impunha, e transpassar essas informações a demais softwares com diferentes linguagens

e suas particularidades.

No estudo de caso abordado neste trabalho, os dados de previsão do tempo são acessados diretamente da base de dados do Mosaico para o público externo em um formato amigável e manipulável, permitindo o desenvolvimento de serviços dos mais variados tipos, sendo que as formas do conteúdo retornável da API, como o JSON, por exemplo, são consumidos de forma praticamente universal. Como tais dados já estão carregados em memória, seu retorno se torna ágil.

Posteriormente, os testes poderão ser ampliados a novas possibilidades, como a utilização de outras bases de dados (datasets), além de lançar e testar essa funcionalidade em um servidor WEB, para utilizar a solução em larga escala. Ademais, a comparação com demais ferramentas, que também possibilitam o uso de APIs, poderá situar as diferenças existentes em relação a eficiência, redigibilidade e legibilidade, fatores chave na decisão por qual ferramenta utilizar no desenvolvimento de um software.

6. Referências

Santos, Wendell. ProgrammableWeb API Directory Eclipses 17,000 as API Economy Continues Surge. ProgrammableWeb, 2017. URL https://www.programmableweb.com/news/programmableweb-api-directory-eclipses-17000-api-economy-continues-surge/research/2017/03/13.

Zavadniak, Cléber. Como funciona uma requisição HTTP. Medium, 2017. URL https://medium.com/clebertech/como-funciona-uma-requisi%C3%A7%C3%A3o-http-cf76f66fe36e.

R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Winston Chang, Joe Cheng, JJ Allaire, Yihui Xie and Jonathan McPherson (2019). shiny: Web Application Framework for R. R package version 1.3.2. https://CRAN.R-project.org/package=shiny

Trestle Technology, LLC (2017). plumber: An API Generator for R. https://www.rplumber.io (site) https://github.com/trestletech/plumber (dev).

RStudio Team (2015). RStudio: Integrated Development for R. RStudio, Inc., Boston, MA. URL http://www.rstudio.com/.

Mesinger, Fedor ; Veljovic, Katarina ; Chou, Sin Chan ; GOMES, JORGE ; LYRA, ANDRÉ . The Eta Model: Design, Use, and Added Value. Topics in Climate Modeling. 1ed.Croacia: InTech, 2016, v. 1, p. 1-16.

Mesinger, F. ; Mesinger, Fedor ; Chou, Sin Chan ; Gomes, J. L. ; Jovic, D. ; BUSTAMANTE, J. F. ; CHOU, Sin Chan ; Lyra, André A. ; LYRA, Andre de Arruda ; Gomes, Jorge L. ; Bastos, Paulo ; Bustamante, Josiane F. ; Veljovic, Katarina ; Jovic, Dusan ; Lazic, Lazar ; Morelli, Sandra ; Ristic, Ivan . An upgraded version of the Eta model. METEOROLOGY AND ATMOSPHERIC PHYSICS, v. 116, p. 63-79, 2012.

Chou, Sin Chan; BUSTAMANTE, J. ; GOMES, J. L. . Evaluation of Eta Model seasonal precipitation forecasts over South America. Nonlinear Processes in Geophysics, Alemanha, v. 12, p. 537-555, 2005.

Roger Bivand, Tim Keitt and Barry Rowlingson (2019). rgdal: Bindings for the 'Geospatial' Data Abstraction Library. R package version 1.4-4. https://CRAN.R-project.org/package=rgdal

Roger S. Bivand, Edzer Pebesma, Virgilio Gomez-Rubio, 2013. Applied spatial data analysis with R, Second edition. Springer, NY. http://www.asdar-book.org/

Robert J. Hijmans (2019). raster: Geographic Data Analysis and Modeling. R package version 2.9-5. https://CRAN.R-project.org/package=raster

H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016.