Sharing your data science work:

Transform your R code into an API with plumber

Gustavo Martinez
Francesca Vitalini
Gabriel Foix

An initiative by
Mirai Solutions GmbH
labs@mirai-solutions.com
https://mirai-solutions.ch

Bringing ideas to life.

Smart.
Agile.
Personal.

MiraiLabs

MiraiLabs: What is it about?

- Are you a data scientist keen on trying new things?
- Do you use data analysis in your daily work? Do you want to expand your toolkit?
- Are you looking for a more guided hands-on introduction?

Data science workshops for professionals

Data science

Data analytics
New tools & techniques
Interactivity & visualization

For professionals

Based on real experience from the industry
Addressing relevant topics

Goals

Learn together
Establish a community

Today’s Outline:

- Renku Platform
- Introduction to APIs
- Introduction to plumber
    - Try it yourself section
- Break ~ 10 min
- Plumber by example
    - Try it yourself section
- Best practices and security

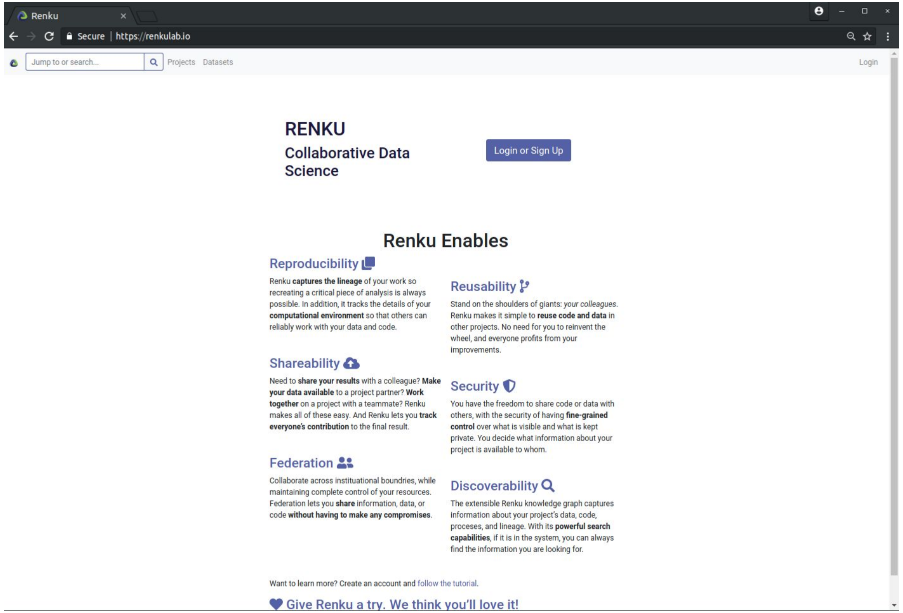
Renku Platform

Renku Platform

Renku is a software platform for data science, developed by the Swiss Data Science Center.

The platform is designed to enable reproducible and collaborative data science.

It is an open source project, and also available as a managed service on the cloud, free to use at https://renkulab.io/



RENKU

Collaborative Data Science

Renku Enables

Reproducibility

Renku **captures the lineage** of your work so recreating a critical piece of analysis is always possible. In addition, it tracks the details of your **computational environment** so that others can reliably work with your data and code.

Reusability

Stand on the shoulders of giants: *your colleagues*. Renku makes it simple to **reuse code and data** in other projects. No need for you to reinvent the wheel, and everyone profits from your improvements.

Shareability

Need to **share your results** with a colleague? **Make your data available** to a project partner? **Work together** on a project with a teammate? Renku makes all of these easy. And Renku lets you **track everyone's contribution** to the final result.

Security

You have the freedom to share code or data with others, with the security of having **fine-grained control** over what is visible and what is kept private. You decide what information about your project is available to whom.

Federation

Collaborate across instituational boundries, while maintaining complete control of your resources. Federation lets you **share** information, data, or code **without having to make any compromises**.

Discoverability

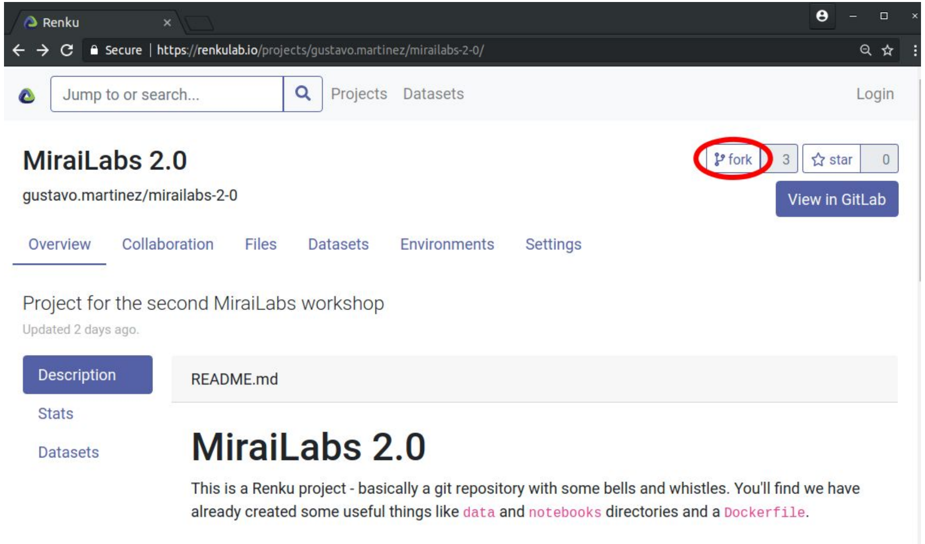
The extensible Renku knowledge graph captures information about your project's data, code, proceses, and lineage. With its **powerful search capabilities**, if it is in the system, you can always find the information you are looking for.

Want to learn more? Create an account and follow the tutorial.

Give Renku a try. We think you'll love it!

MiraiLabs on Renku

1.- Open the MiraiLabs 2.0 project and fork it.



Renku

Secure | https://renkulab.io/projects/gustavo.martinez/mirailabs-2-0/

Jump to or search... Projects Datasets Login

MiraiLabs 2.0

gustavo.martinez/mirailabs-2-0

fork 3 star 0

View in GitLab

Overview Collaboration Files Datasets Environments Settings

Project for the second MiraiLabs workshop

Updated 2 days ago.

Description

Stats

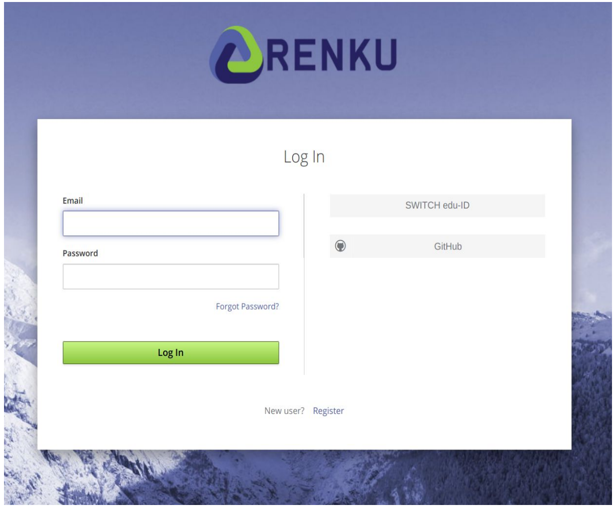
Datasets

README.md

MiraiLabs 2.0

This is a Renku project - basically a git repository with some bells and whistles. You'll find we have already created some useful things like `data` and `notebooks` directories and a `Dockerfile`.

2.- Log in or register as a new user. You can use your GitHub account to log in.



RENKU

Log In

Email

Password

Forgot Password?

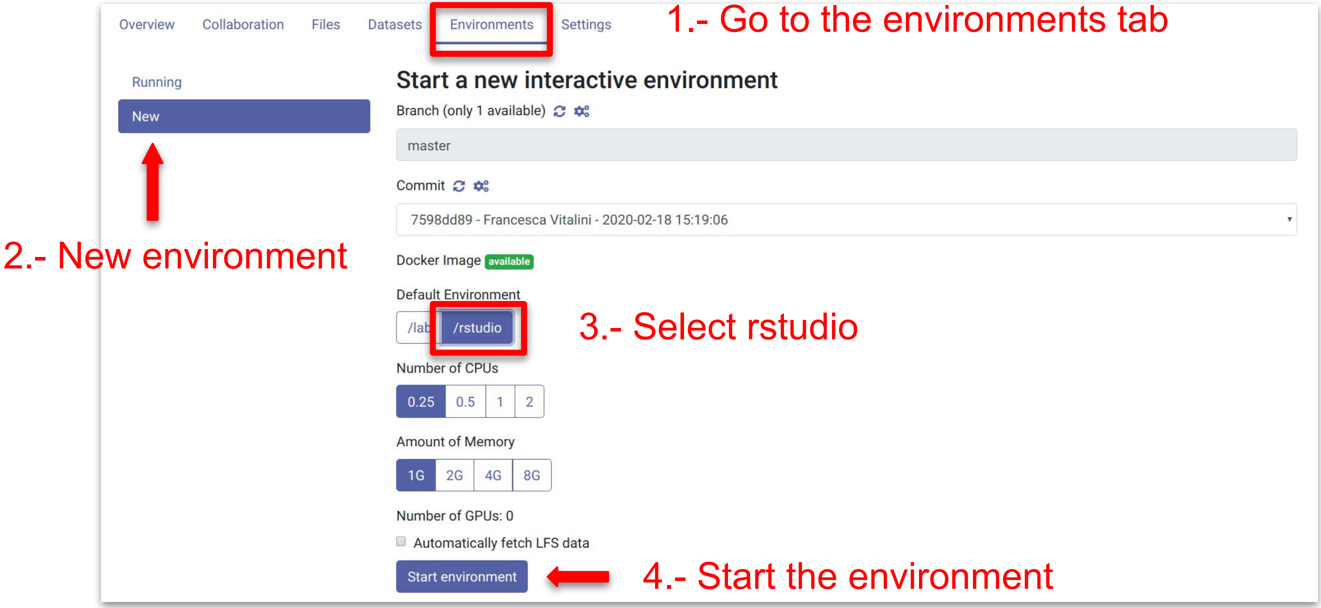
Log In

SWITCH edu-ID

GitHub

New user? Register

Spin-up your own environment



Overview | Collaboration | Files | Datasets | Environments | Settings

1.- Go to the environments tab

Running

New

2.- New environment

Start a new interactive environment

Branch (only 1 available)

master

Commit

7598dd89 - Francesca Vitalini - 2020-02-18 15:19:06

Docker Image available

Default Environment

/lab | /rstudio

3.- Select rstudio

Number of CPUs

0.25 | 0.5 | 1 | 2

Amount of Memory

1G | 2G | 4G | 8G

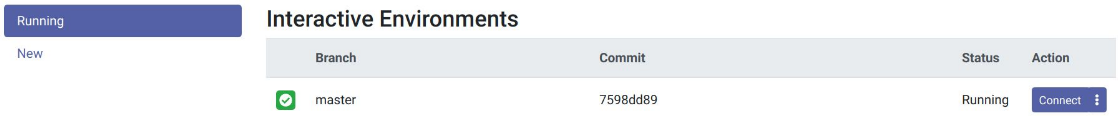
Number of GPUs: 0

□ Automatically fetch LFS data

Start environment

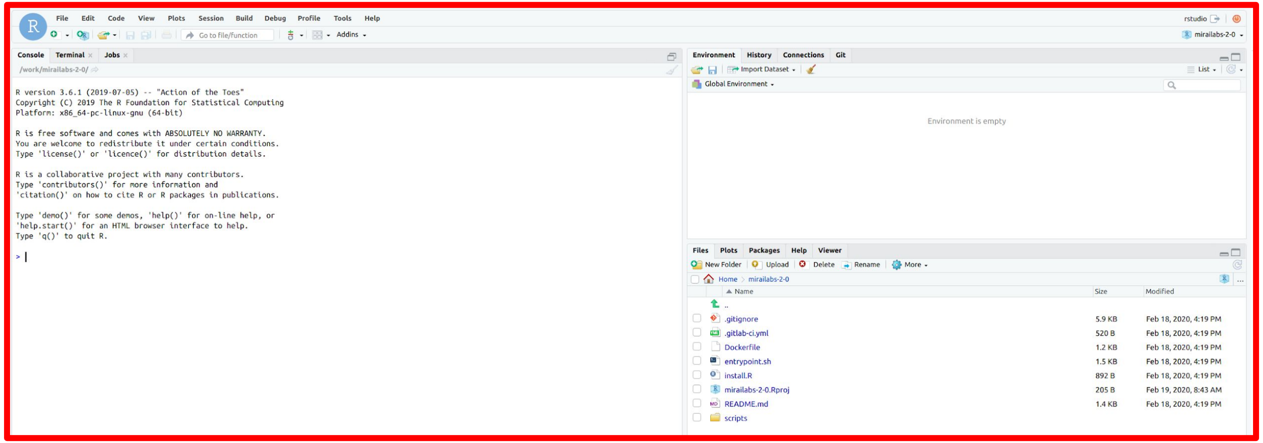
4.- Start the environment

Connect to your environment



Interactive Environments

	Branch	Commit	Status	Action
✓	master	7598dd89	Running	Connect ⋮



/work/mirailabs-2-0/

```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Name	Size	Modified
..		
.gitignore	5.9 KB	Feb 18, 2020, 4:19 PM
.gitlab-ci.yml	520 B	Feb 18, 2020, 4:19 PM
Dockerfile	1.2 KB	Feb 18, 2020, 4:19 PM
entrypoint.sh	1.5 KB	Feb 18, 2020, 4:19 PM
install.R	892 B	Feb 18, 2020, 4:19 PM
mirailabs-2-0.Rproj	205 B	Feb 19, 2020, 8:43 AM
README.md	1.4 KB	Feb 18, 2020, 4:19 PM
scripts		

Connect to open up your RStudio session. It will open up the workshop project, with all necessary scripts and dependencies already available/

Web APIs

Web APIs

Web APIs (Application Programming Interface) are communication protocols between computer programs connected in a network.

Unnoticed most of the time, they are the bread and butter of digital life.

Data scientists often use them for data sourcing, often through supporting tools.

HTTP vs REST

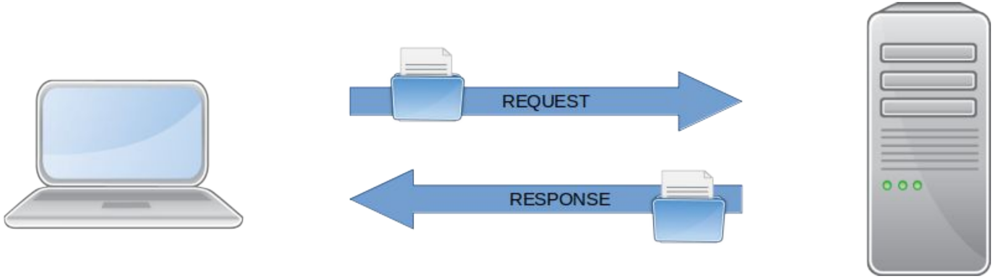
HTTP (Hypertext Transfer Protocol) is a communication protocol for transferring information through a network. It is the most common for web APIs. HTTPS is a secure form of HTTP.

REST (REpresentational State Transfer) is a software architectural style for web APIs. REST defines constraints on the software design, not on the technology.

A REST API does not necessarily need to use HTTP.

HTTP

HTTP



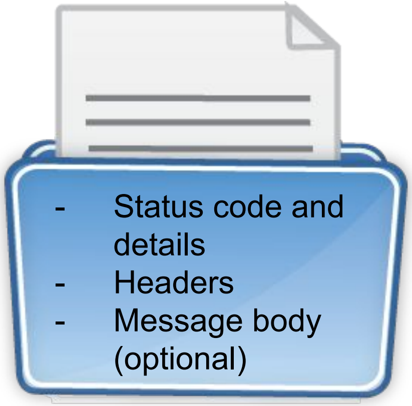
A client sends a request message to a server. The server interprets the request and sends back a response.

HTTP messages are packets of information for which the protocol provides a uniform interface.

HTTP messages

- Request URL
- HTTP method
- Headers
- Message body (optional)

Parts of the request message



- Status code and details
- Headers
- Message body (optional)

Parts of the response message

HTTP message parts

HTTP

- **URL**[1]. The address or endpoint for the request. Can contain parameters.
- **HTTP method**[2]. A specific process invoked on the endpoint.
- **Headers**. Additional information such as who is making the request, what type of response is expected, or format of the message body.
- **Message body**[3]. Actual data transferred with the message. Serialized.

Commonly used terms referring to the same concepts: [1]URI, [2]Verb, [3]Payload

HTTP methods

HTTP

Method	Operation	Description	Status codes
GET	Read	Retrieve a representation of a resource.	200 (OK). Common errors: 404 (NOT FOUND), 400 (BAD REQUEST)
POST	Create	Create new resources, typically subordinated to other (e.g. parent) resource	201 (Created). 409 (Conflict)
PUT	Update / replace	Update a known resource. Sometimes create, if the id of the resource is decided by the client.	200 (OK), 204 (No Content). 404 (Not Found)
DELETE	Delete	Delete a resource	200 (OK). 404 (Not Found), 405 (Method Not Allowed)

Most common methods, used for CRUD operations (Create, Read, Update, Delete)

HTTP status codes

HTTP

CATEGORY	DESCRIPTION
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client’s request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.

See more detailed lists in Wikipedia or restapitutorial

Web clients

There are multiple options for web clients that can be used to place calls to web APIs and interpret their responses:

- Web browsers addons
- Standalone clients, like Postman or Insomnia
- Swagger UI. Interactive UI tool, the choice of RStudio
- Command line tool curl

Web APIs in R

API wrappers

Many R packages include API wrappers: user-friendly functions for interacting with web APIs from within R.

The wrappers do all the heavy lifting work for us: defining the HTTP requests, and parsing the responses.

For example, the package `tidyquant` integrates resources for collecting financial data from various sources.

Other examples are `bigrquery`, `worldmet` the `cloudyr` project, `...`

R web tools

There are multiple packages implementing web clients based on libcurl. We will see some **`httr`** examples (`01-intro_web_APIs.R`).

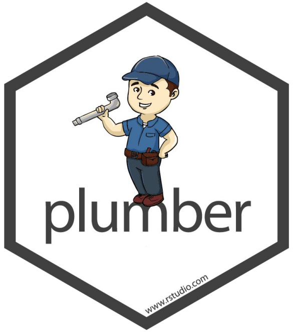
`urltools` provides useful tools for handling urls.

`plumber` is a friendly API generator that uses `httpuv`, a low level web server library.

In this workshop we will focus on `plumber,` which is already available in your **Renku** project.

PlumbeR

Plumber



`plumber` is an open source R package from RStudio to build and run web APIs. With plumber, you can convert existing R code into a web API, by decorating it with special comments.

Get `plumber`:

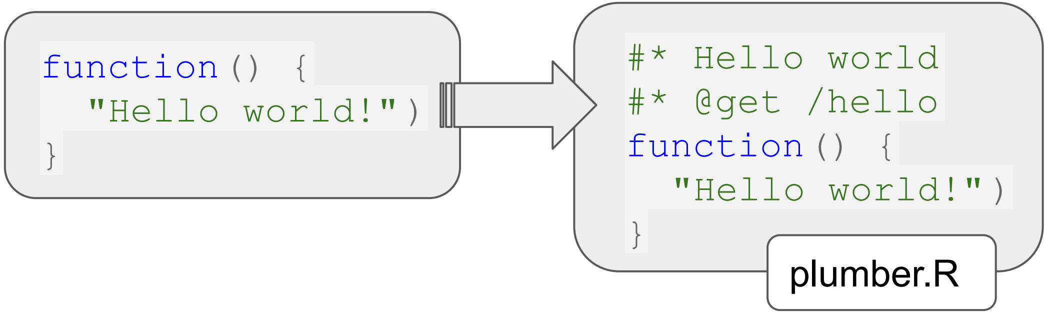
```
install.packages(“plumber”)
```

“`plumber`-izing” your R code will make your decorated functions available as API endpoints.

Using plumber: decorating your R code

```
function() {
  "Hello world!")
}
```

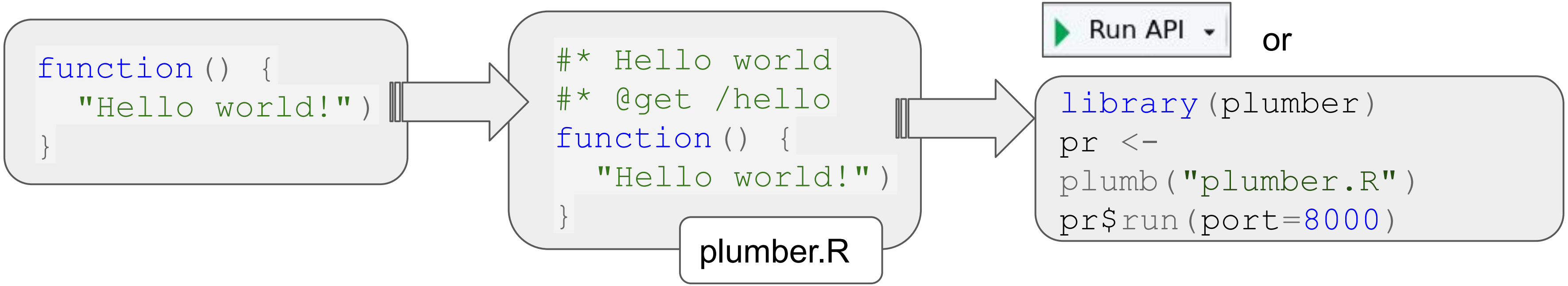
Using plumber: decorating your R code



```r
function() {
  "Hello world!")
}
```

```r
#* Hello world
#* @get /hello
function() {
  "Hello world!")
}
```

Using plumber: decorating your R code



```
function() {
  "Hello world!")
}
```

```
#* Hello world
#* @get /hello
function() {
  "Hello world!")
}
```

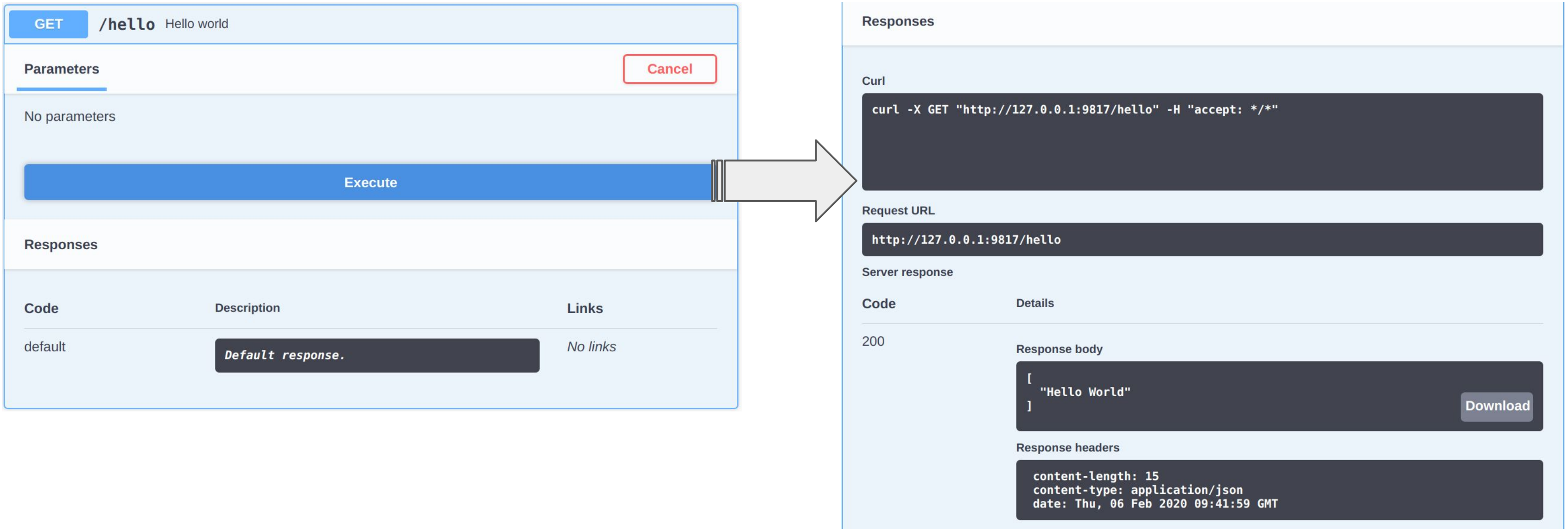
plumber.R

Run API

or

```
library(plumber)
pr <-
plumb("plumber.R")
pr$run(port=8000)
```

Using plumber: Swagger UI



`/hello` Hello world

Parameters

No parameters

Responses

Code	Description	Links
default	*Default response.*	*No links*

Responses

Curl

```
curl -X GET "http://127.0.0.1:9817/hello" -H "accept: */*"
```

Request URL

```
http://127.0.0.1:9817/hello
```

Server response

Code	Details
200	Response body

```
[
  "Hello World"
]
```

Response headers

```
content-length: 15
content-type: application/json
date: Thu, 06 Feb 2020 09:41:59 GMT
```

Using plumber: serializing the response

By default plumber serializes the response in json format. Other content-types can be specified through annotations:

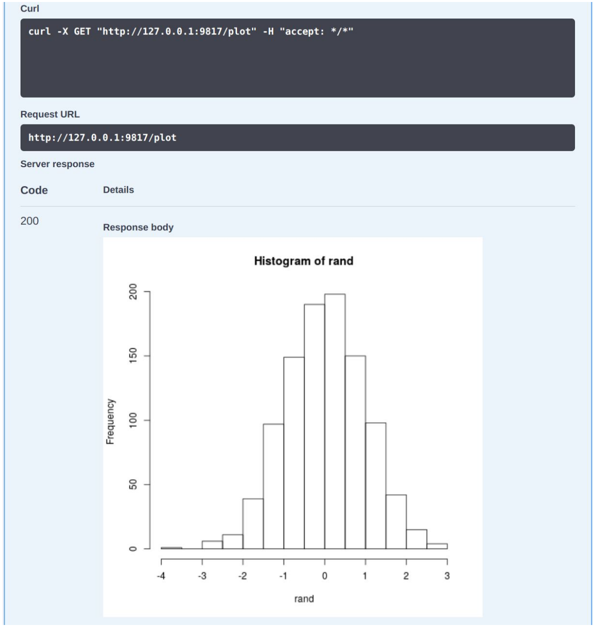
Annotation	Content Type	Description/References
`@json`	`application/json`	`jsonlite::toJSON()`
`@html`	`text/html; charset=utf-8`	Passes response through without any additional serialization
`@jpeg`	`image/jpeg`	`jpeg()`
`@png`	`image/png`	`png()`

Using plumber: return an image

```
#* Plot a histogram
#* @png
#* @get /plot
function() {
  rand <- rnorm(1000)
  hist(rand)
}
```

The serializer tag `#* @png` ensures that the output is correctly interpreted as an image.

Other formats are `#* @html` or `#* @jpeg`; the default is `#* @json`.



Curl

```
curl -X GET "http://127.0.0.1:9817/plot" -H "accept: */*"
```

Request URL

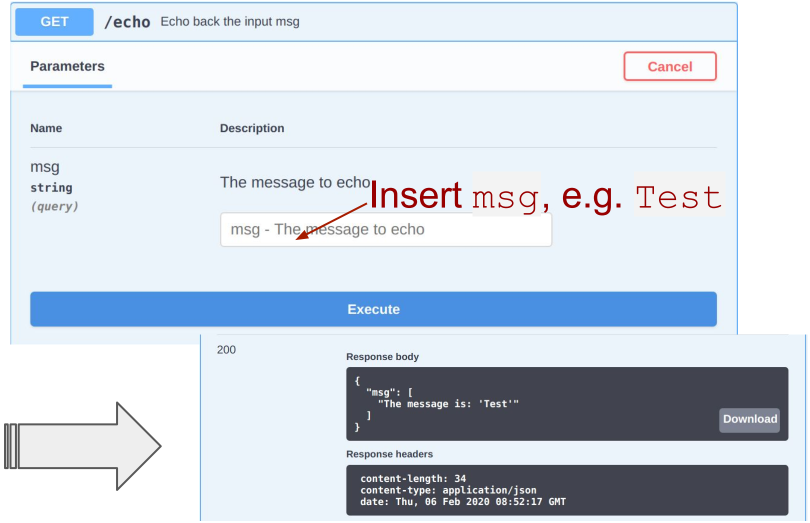
```
http://127.0.0.1:9817/plot
```

Server response

Code	Details
200	Response body

Using plumber: passing parameters

```
#* Echo back the input msg
#* @param msg The message to echo
#* @get /echo
function(msg = "") {
  list(msg = paste0("The message
is: '", msg, "'"))
}
```



Insert `msg`, e.g. `Test`

```
{
  "msg": [
    "The message is: 'Test'"
  ]
}
```

```
content-length: 34
content-type: application/json
date: Thu, 06 Feb 2020 08:52:17 GMT
```

Request URL with parameter in the query string

`http://127.0.0.1:9817/echo?msg=Test`

Curl

`curl -X GET "http://127.0.0.1:9817/echo?msg=Test" -H "accept: */*"`

Using plumber: dynamic routes

```
#* Plot out data from the iris dataset.
Dynamic route (e.g. 'setosa',
'virginica')
#* @param species Filter the data to get
only this species (e.g. 'setosa')
#* @get /iris/<species>
#* @png

function(species) {
...
```

In a dynamic route, the URL path contains parameters.

The URL combines static parts and parameter values, separated by “/”

Request URL example:

`http://127.0.0.1:9817/iris/` **`setosa`**

Using plumber: typed dynamic routes

```
#* Return the sum of two numbers.
Typed dynamic route.
#* @param a The first number to add
#* @param b The second number to add
#* @get /sum/<a:numeric>/<b:numeric>
function(a,b) {
  a + b
}
```

Parameters in typed dynamic routes are the only parameters whose type is controlled by plumber.

Request URL example

`http://127.0.0.1:9817/sum/` **`2/1`**

R Type	Plumber Name
logical	`bool`, `logical`
numeric	`double`, `numeric`
integer	`int`

Using plumber: debugging

Printing messages to track progress:

```
...
print(paste0("a is of class: ", class(a)))
...
```

Interrupt execution for inspection (browser), or debug a function (debug). Only possible when running interactively:

```
...
browser()
...
```

```
...
debug(myfunction)
...
```

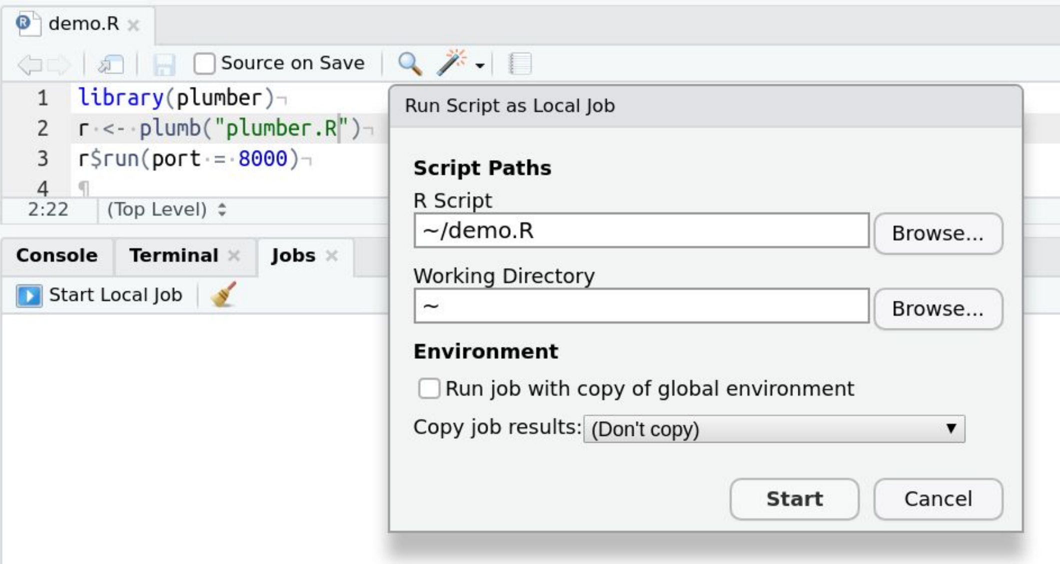
Running plumber as a job in RStudio

Running a `plumber` API as a local job in RStudio, to not have the R session busy.

Multiple jobs can serve multiple APIs at once, in different ports.

```r
library(plumber)
pr <-
plumb("plumber.R")
pr$run(port=8000)
```

demo.R



```r
library(plumber)
r <- plumb("plumber.R")
r$run(port = 8000)
```

Run Script as Local Job

Script Paths

R Script: ~/demo.R

Working Directory: ~

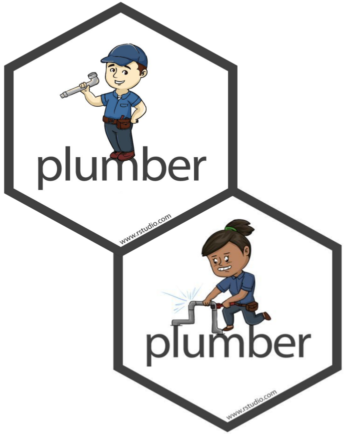
Environment

□ Run job with copy of global environment

Copy job results: (Don't copy)

Plumber resources

- Github: https://github.com/rstudio/plumber
- Plumber documentation https://www.rplumber.io/docs/
- Webminar https://rstudio.com/resources/videos/plumbing-apis-with-plumber/



Try it yourself

- `GET` current date and time (`Sys.time()`)
- Decorate a function that would return the sentence `"Today is … "` based on the input `day`.
- Consider the function to return the predicted miles per gallon for a given cylinder:

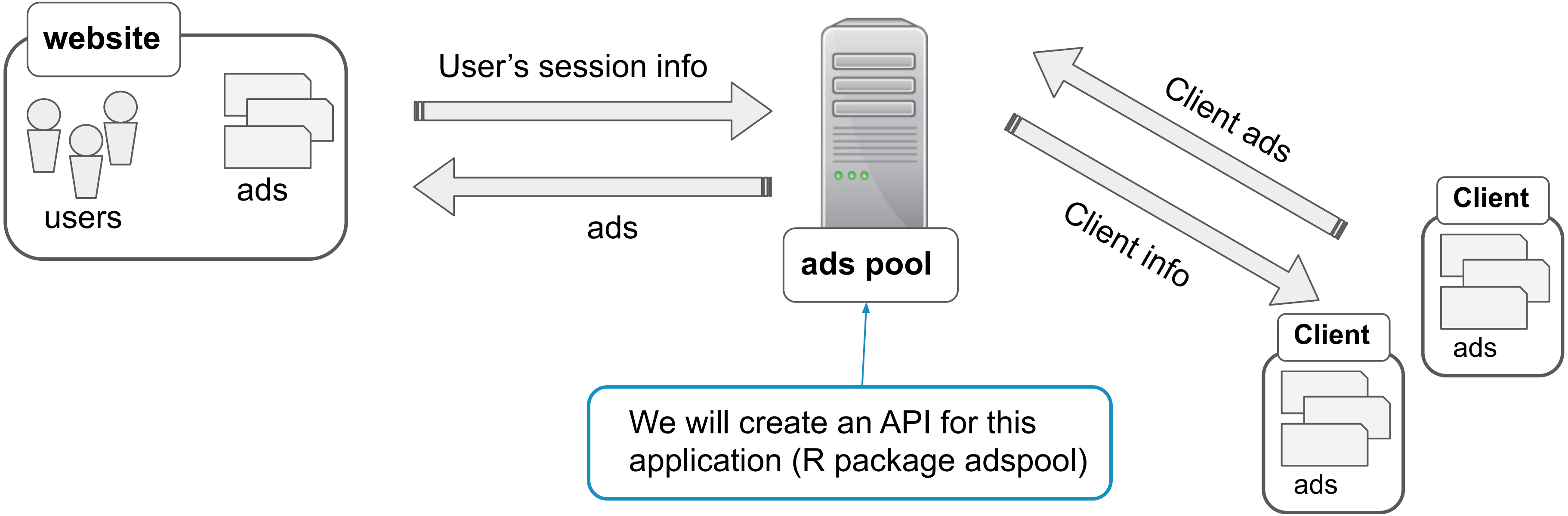
```
function(cyl) {
  predict(lm(mpg ~ cyl, data = mtcars), data.frame(cyl = cyl))
}
```

Make a `GET` endpoint to serve the model (hint: what type should `cyl` have for this to work?)

Plumber

“by example”

The “adverts pool” example



We will create an API for this application (R package adspool)

The adspool dataset

The `R` package, `adspool`, generates the `ads` dataset:

id	name	category	subcategory	img_path	customer_id	click_count	click_rate
1	Mario Kart	Games	Videogame	001_MarioKart.jpeg	10	113	0.02
2	Fifa	Games	Videogame	002_Fifa.jpeg	7	254	0.02
3	Assassin's Creed	Games	Videogame	003_AssassinsCreed.jpeg	6	186	0.03
4	Fortnite	Games	Videogame	004_Fortnite.jpeg	3	120	0.01
5	The Sims	Games	Videogame	005_TheSims.jpeg	9	90	0.02

The adspool tools

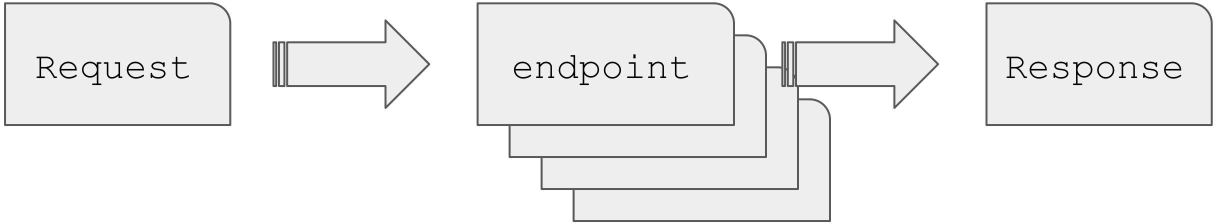
The `R` package, `adspool`, provides tools to:

- Select the best advertisement in a given input context (`select_ads()`)
- Return a subset of advertisements based on input parameters (`subset_ads()`)
- Create an advertisement in the ads pool (`add_ad()`)
- Modify an advertisement in the ads pool (`update_ad()`)
- Remove an advertisement from the ads pool (`remove_ads()`)
- Produce an open invoice for a given customer

The adspool API endpoints

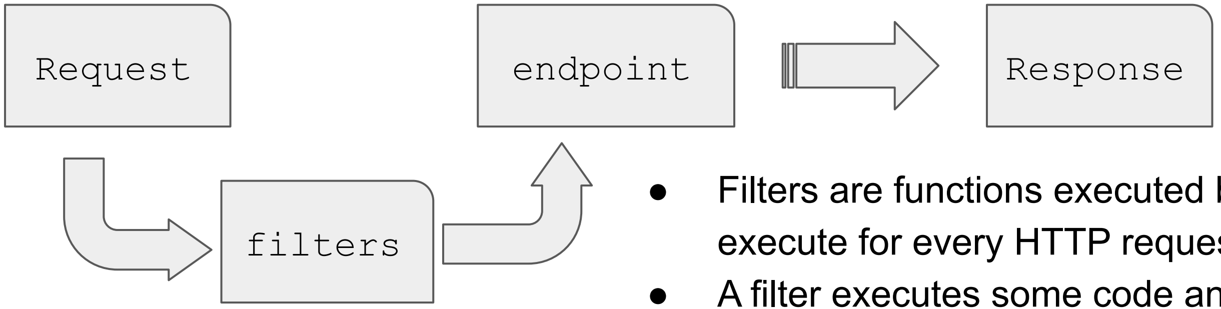
- Get advertisements data for a given name, category, subcategory etc.
- Return all advertisements of a given customer
- Get advertisements profit graph
- Get advertisement image
- Get customer open invoice
- Return ads as R `rds` object
- Delete advertisement from ads pool
- Create a new advertisement in ads pool
- Modify an existing advertisement
- Get advertisements using metadata from a cookie

Plumber routers



- Plumber executer R code in response to HTTP requests.
- Incoming HTTP requests must be routed to one or more R functions.
- Plumber will try to find one (and only one) endpoint with an R function to execute.

Plumber routers



- Filters are functions executed before the endpoint. All filters execute for every HTTP request.
- A filter executes some code and then either returns a response (interrupting execution) or forwards to the next handler (filter or endpoint)
- Both filters and endpoints have access to the request and response objects

The Request object (`req`)

The Request Object in `plumber` is an environment that satisfies the Rooker Interface. `req` encapsulates all the details of the client's request, including:

- Request Path & Method
- Query String, the part of the `URL` after `?`.
- Request Body, called as `req$postBody` (but not exclusive of post methods).
- Cookies, as `req$cookies`.
- HTTP Headers, attached to the request as `req$HTTP_HEADER_NAME`.

The Response object (`res`)

The `plumber` response object `res` is an object of a built-in class, `PlumberResponse`

`res` encapsulates all the information of the response sent back to the client, including:

- Status code
- Body, serialized as per endpoint specification
- Headers

The response object includes some internal methods, notoriously `setCookie`

Plumber cookies

Store information about the state, in a stateless API

```
#* Capture users behavior on a cookie
#* @param usersession
#* @put /ads/usersession/
function(res, usersession) {
  res$setCookie("usersession", usersession)
}
```

`usersession` information available to other endpoints as `req$cookies$usersession`.

Application Examples

- Get advertisements based on browsing history passed as a request body
- Mimic a click on an advertisement
- Get advertisements based on user session (using cookies)
- Log information with a filter logger
- Handle error messages using a filter
- Return a static file

Try it yourself:

Using the `adspool` package as a starting point:

- Write an endpoint to GET all the advertisements belonging to one subcategory. (hint: check `get_customer_ads`)
- Set a `cookie` that will store the time at which the endpoint `get_time` was requested (hint: use `Sys.time()`, check `get_counter()`)
- Create a `filter` that will print to the console the time at which the endpoint was requested (hint: use `Sys.time()`, check `filter_logger()`)

Plumber Advanced

Programmatic usage of plumber

- Creating a router: `pr <- plumber$new()`, object of the class `Plumber`
- Adding endpoints with the method `handle`:

```
pr$handle(methods = "POST",
          path = "string",
          handler = function() {},
          params = list(),
         serializer = serializer())
```

Add filters and register hooks

- Add filters with the `filter` method:

```
pr$filter(name = "filter name",
          expr = expiration_in_secs,
        serializer = serializer())
```

- Define “hooks” to execute code at a particular point of the request’s lifecycle with the `registerHook(s)` method(s). Currently supported hooks: `preroute, postroute, preserialize, postserialize`

Mounting routers and static files

- Mounting routers with the method `mount`:

  ```
  pr$mount("/path", other_router)
  ```

  Allows modularisation and compartmentalisation of APIs
- Routers can be mounted on top of each other, creating a tree of paths and endpoints
- Static file servers can be combined with routers. These are objects of the class `PlumberStatic`:

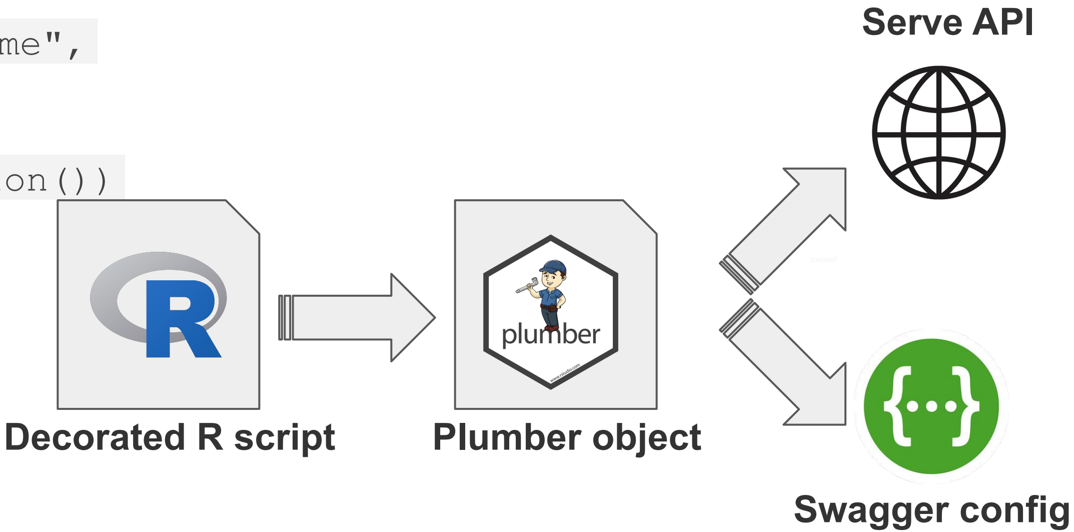
  ```
  stat <- PlumberStatic$new("./myfiles")
  pr$mount("/assets", stat)
  ```

Running plumber

- Serve your API with the `run` method:

```
pr$run(host = "host name",
       port = 8000,
       swagger = function())
```

All parameters are optional.



Plumber limitations

- HTTPS not supported natively
- No built-in authentication and no support for `oauth`
- Some common serialization formats are not built-in (e.g. `xml`, `multipart/format`), although one can define custom serializers
- Files upload through endpoint currently under development
- OpenAPI specs are not always complete. Notoriously: requestBody missing
- Functionality exposed through decorators may be limited sometimes

Serving Plumber

- Hosting platforms
  - DigitalOcean
  - RStudio Connect
- Hosting on self-managed server
  - Docker

Serving Plumber. Hosting platforms

- DigitalOcean

  Online, easy to use cloud service, with a small payment plan based on usage. Plumber provides an easy way to deploy an API there (`plumber::do_deploy_api()`)

- RStudio Connect

  Enterprise publishing platform from RStudio. Requires a Licence. It offers buttons to deploy various applications, including APIs. More integration is to be expected as main author of `plumber` also works on RStudio Connect.

Serving Plumber. Self-managed.

A plumber API service is simply a process running an R script. Docker containers provide a convenient way to manage dependencies and ship the final product

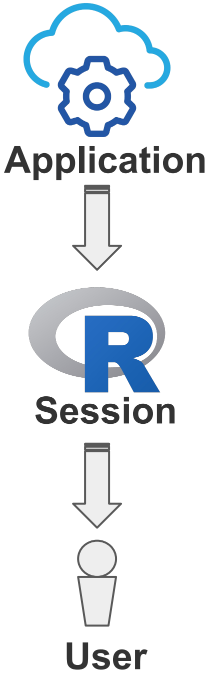
- Docker

  Docker image available for plumber: `docker pull trestletech/plumber`

  Run your `plumber.R` script (assuming no dependency and that the command is run from the same folder as the script) and serve the API on `localhost:8000/`

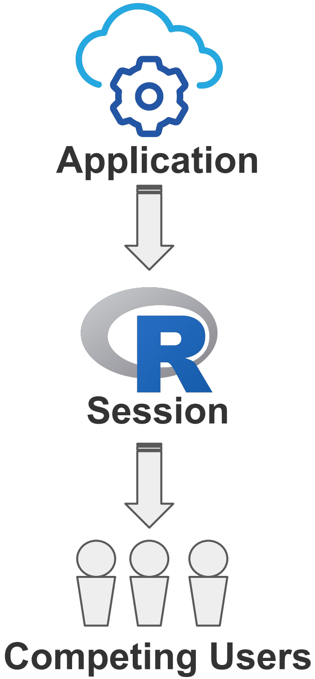
  ```
  docker run --rm -p 8000:8000 -v `pwd`/plumber.R:/plumber.R
  trestletech/plumber /plumber.R
  ```

Serving Plumber



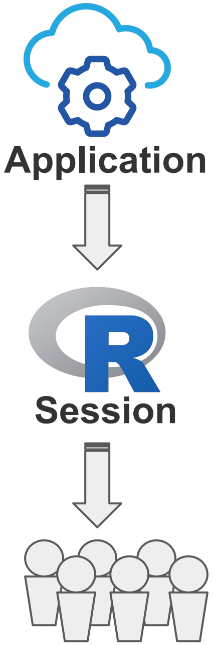
R is single-threaded: one R session serves one R process at a time

Serving Plumber



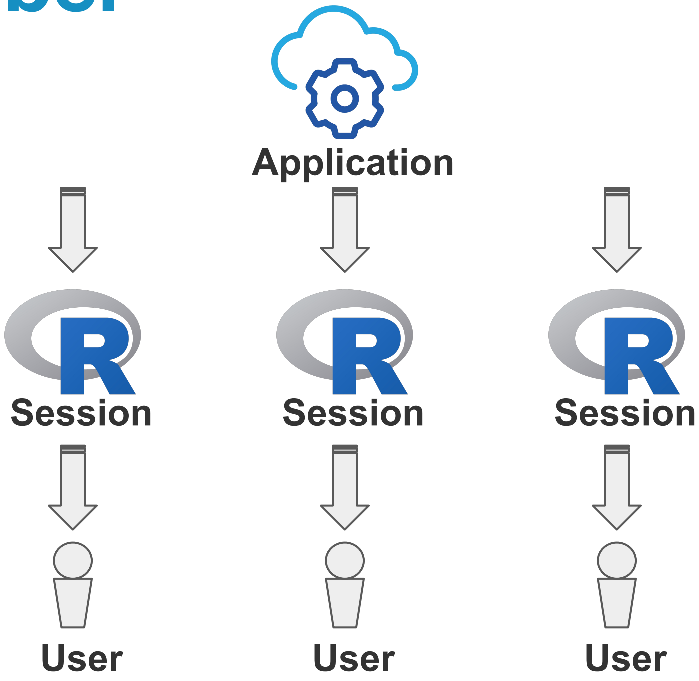
Competing users may queue and have to wait in case of computationally intensive processes

Serving Plumber



Too many Users.
The application
becomes unusable

Serving Plumber



A natural solution is serving the API redundantly

This opens new challenges, like load balancing and ensuring data consistency. Good API design practices become critical

Best Practices

Best Practices and Security

Exposing a web API poses challenges that may be new to many R programmers:

- **Best practices**. Commonly accepted standards, and good documentation become critical, in order for the API to be usable by others.
- **Security**. Requires specialized support on a case-by-base. The R programmer needs to be at least aware of some fundamentals.

Best Practices

- Observe generally accepted recommendations for the choice of methods and status codes.
- Control data types.
- Control errors. Return informative error messages, especially for malformed requests.
- Modularize your code. Use router mounting. Make API versions explicit in the URL.
- Use resource expansion and pagination.
- Be as stateless as possible.
- Consider scalability when designing your API.
- Managing state is challenging, leverage transactional data storages when possible.
- Manage I/O: control your connections and close them when no longer necessary.

Security

Beware of the context in which you will expose your API:

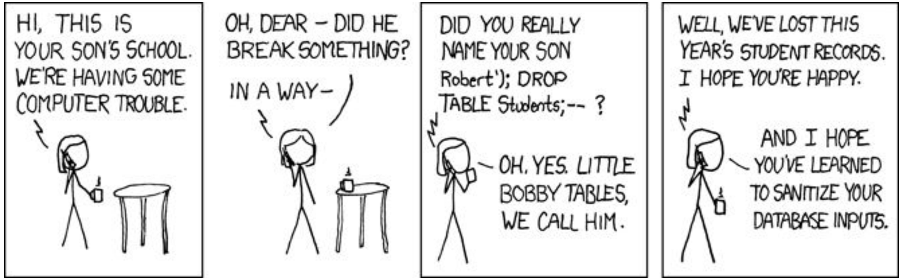
- Network environment & firewalls between you and the server hosting the API.
- HTTPS. `plumber` does not support HTTPS natively, this can be added on top though.
- Authentication does not come for free with `plumber` (or any other such framework).

Get familiar with some ot the basics of security breaches:

- Code injection
- Denial of Service and Distributed Denial of Service (DoS/DDoS)
- Cross-site scripting

Security. Good practices.

- Manage your secrets: do not store keys/passwords in the code or in places of easy access.
- Restrict the rights of the users that execute your services to the minimum required.
- Distinguish between trusted and untrusted objects. All user input is untrusted: Sanitize.
- Be attentive to the amount of resources your application may consume.
- Think in potential malicious usage of your application.


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Q&A

Closing Remarks

MiraiLabs - Next Steps

Feedback welcome:

- Which are the topics you are interested in?
- Format of the workshop: Facilities, materials.

Get in touch!

labs@mirai-solutions.com

Thank you!

Apero Time