

# Mastering Plumber Structure: Your API's Solutions

Adam Foryś, Principal Data Scientist, Roche  
Magdalena Krochmal, Senior Data Scientist, Roche

# Plumber API

Table of contents

**GET**

**/intro** Introduction to APIs

**GET**

**/intro/plumber** About the Plumber API framework

**POST**

**plumber/basic** Basic plumber API implementations



**POST**

**plumber/as\_code** Programmatic usage



**POST**

**plumber/as\_package** Zip the plumber into package



**POST**

**/performance** Enhance API performance future.Promise()

**PUT**

**/select** How to choose the right approach?

**PUT**

**/learn** Learn more

GET

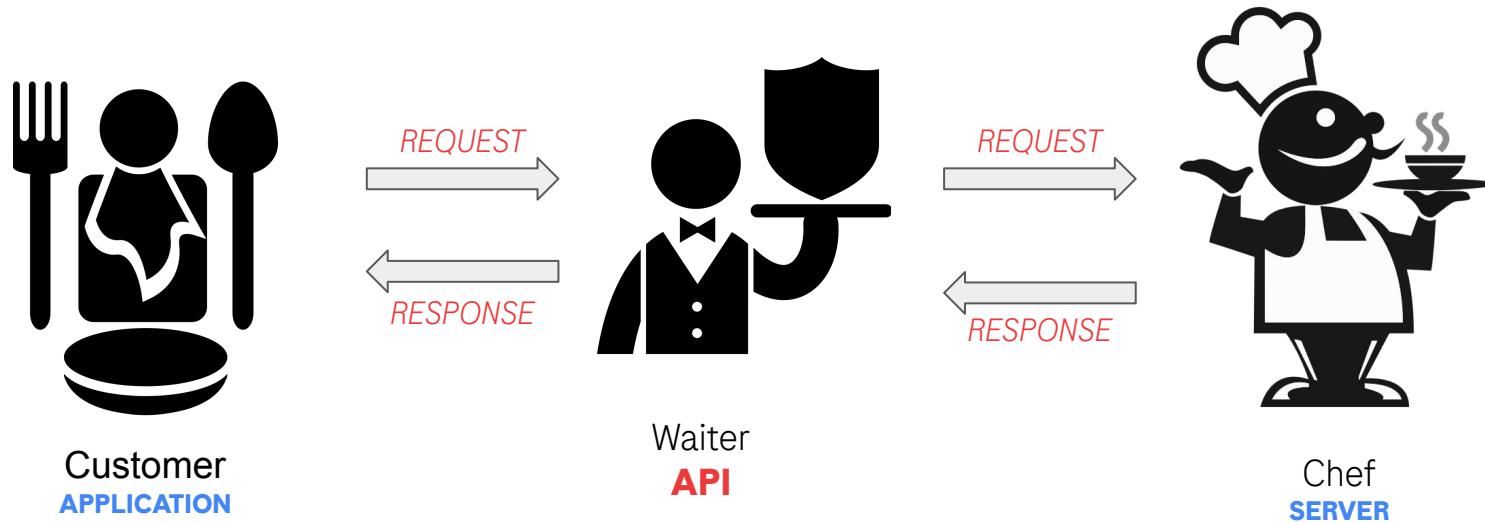
/intro/api Introduction to APIs



# What is an API?

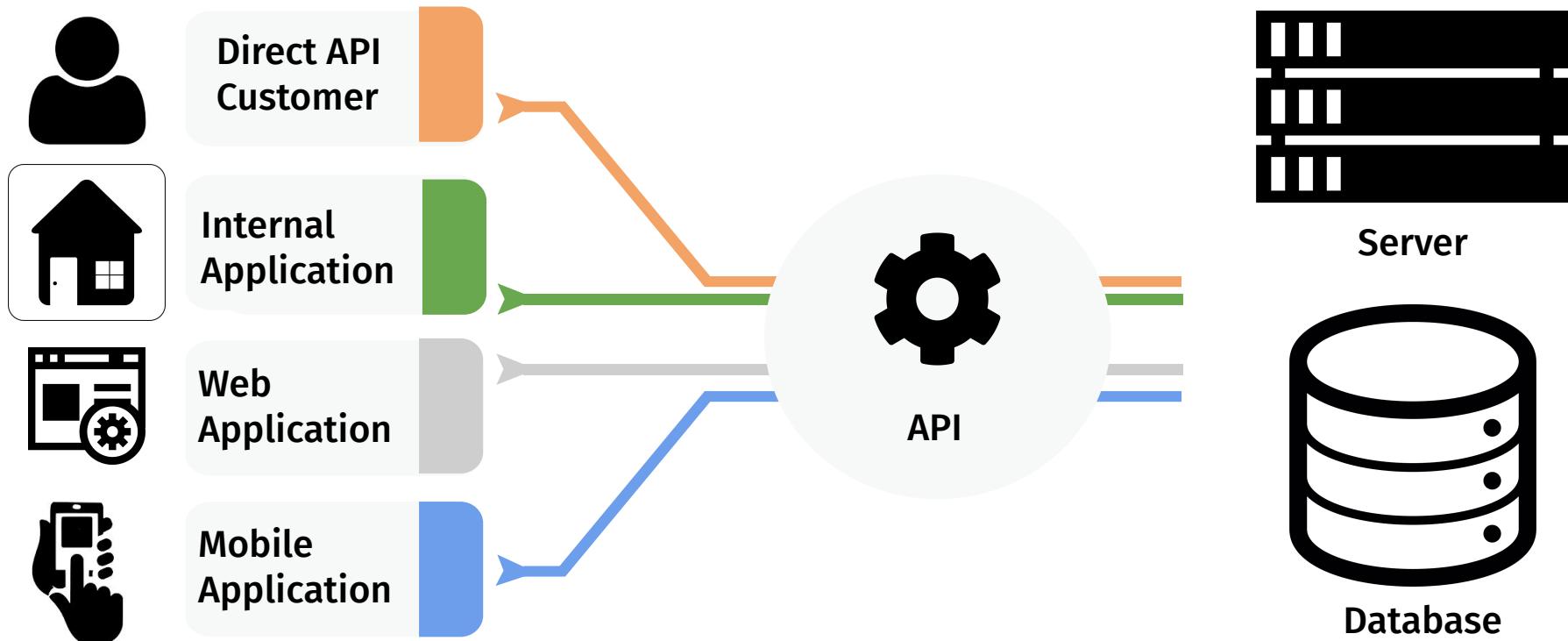
API = Application Programming Interface

Communication channel between two different programs or systems



GET

## /intro/api/types Types of APIs



GET

## /intro/api/examples Everyday examples of APIs



Google  
Maps

Maps API



Grammarly API



PayPal

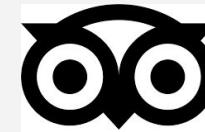
PayPal



Spotify



Dropbox



Tripadvisor

Tripadvisor

GET

/intro/plumber About the Plumber API framework



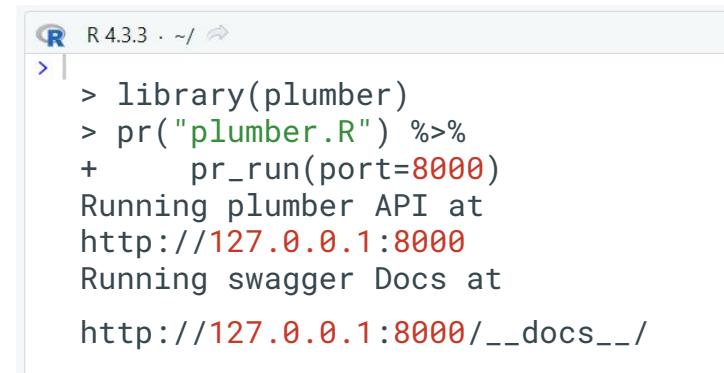
The plumber R package allows users to expose existing R code as a service available to others on the Web.



plumber.R

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

## QUICKSTART

A screenshot of an R console window titled "R 4.3.3". The console shows the following R code being run:

```
> library(plumber)
> pr("plumber.R") %>%
+   pr_run(port=8000)
Running plumber API at
http://127.0.0.1:8000
Running swagger Docs at
http://127.0.0.1:8000/__docs__/
```

GET

/intro/plumber About the Plumber API framework



useR! Salzburg  
2024

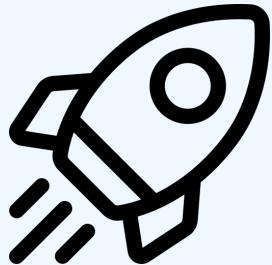
The screenshot shows the Swagger UI interface for an API titled "API Title 1.0.0 OAS3". The URL is https://krochmam-o48ket.eu.aa.ocean.roche.com/s/7a9a56cadd1f30584d3e6/p/db452799/openapi.json. The "Servers" dropdown is set to https://krochmam-o48ket.eu.aa.ocean.roche.com/s/7a9a56cadd1f30584d3e6/p/db452799/. Below it, a "default" dropdown shows a "GET /echo Echo back the input" button.

A browser window displays a JSON response with "Pretty-print" checked. The response is:

```
{  
  "msg": [  
    "The message is: 'Hello, user!'"  
  ]  
}
```

GET

/intro/plumber About the Plumber API framework



## DEPLOYMENT / HOSTING



DigitalOcean

Containers as a Service (CaaS)

Docker

The screenshot shows the RStudio interface with the "plumber.R" file open in the editor. The code defines a simple echo function:

```
1  ## Echo back the input
2  ## @param msg The message to echo
3  ## @get /echo
4  function(msg = "") {
5      list(msg = paste0("The message is: '", msg, "'"))
6  }
```

A context menu is open at the bottom right of the editor, with "Publish API..." highlighted. The "Publishing Accounts" pane in the options menu shows a single account listed. Two blue arrows point from the "Run API" and "Publish API..." buttons in the menu bar down to the corresponding items in the context menu.

Options

- R General
- Code
- Console
- Appearance
- Pane Layout
- Packages
- R Markdown
- Python
- Sweave
- Spelling
- Git/SVN
- Publishing**
- Terminal
- Accessibility

Publishing Accounts

- Connect...
- Reconnect...
- Disconnect

Settings

- Enable publishing documents, apps, and APIs
- Enable publishing to Posit Connect ?
- Enable publishing to Posit Cloud
- Show diagnostic information when publishing

SSL Certificates

- Check SSL certificates when publishing
- Use custom CA bundle

Troubleshooting Deployments

Untitled1\* plumber.R\*

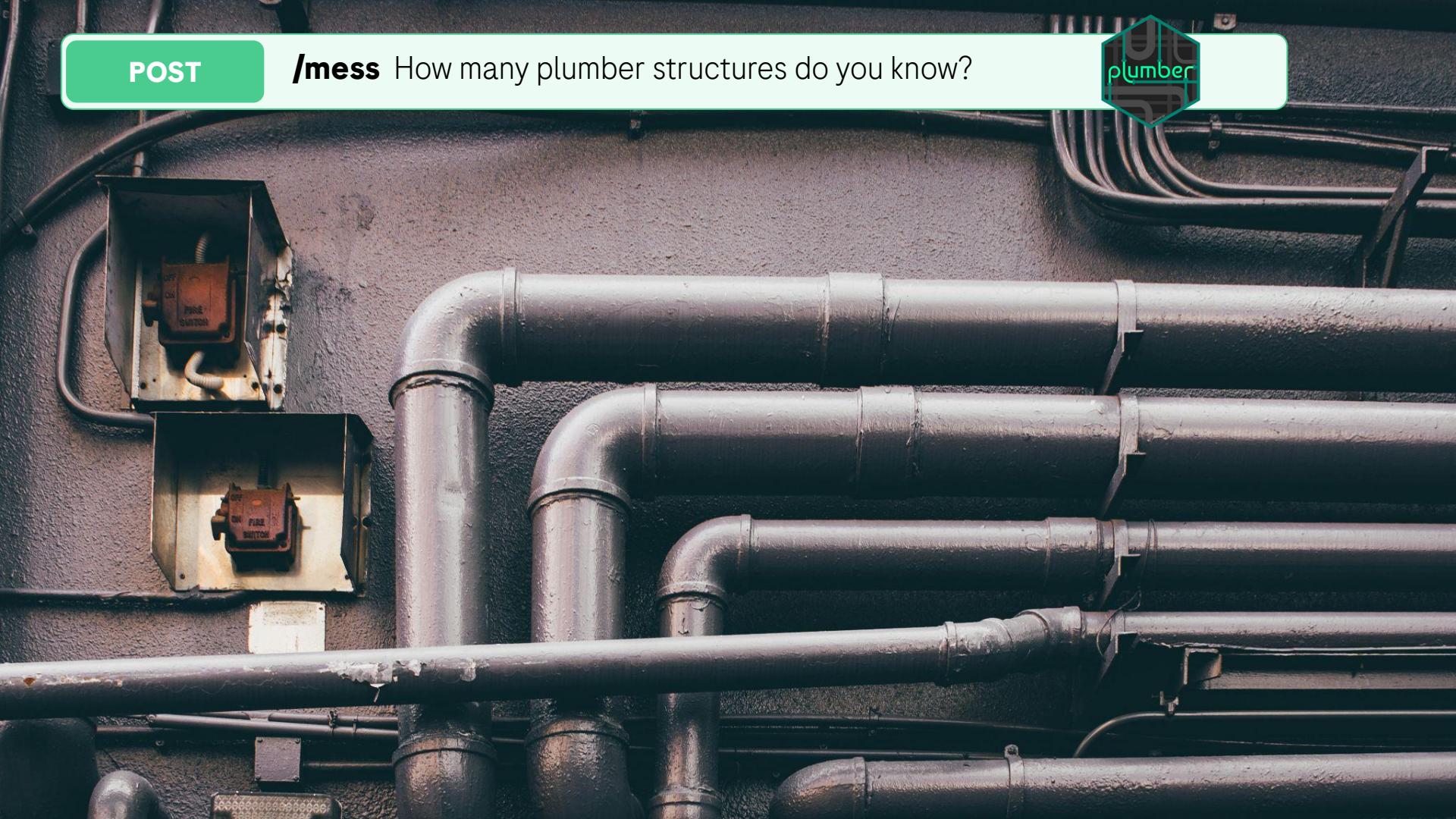
Run API

Publish API...

Manage Accounts...

**POST**

**/mess** How many plumber structures do you know?

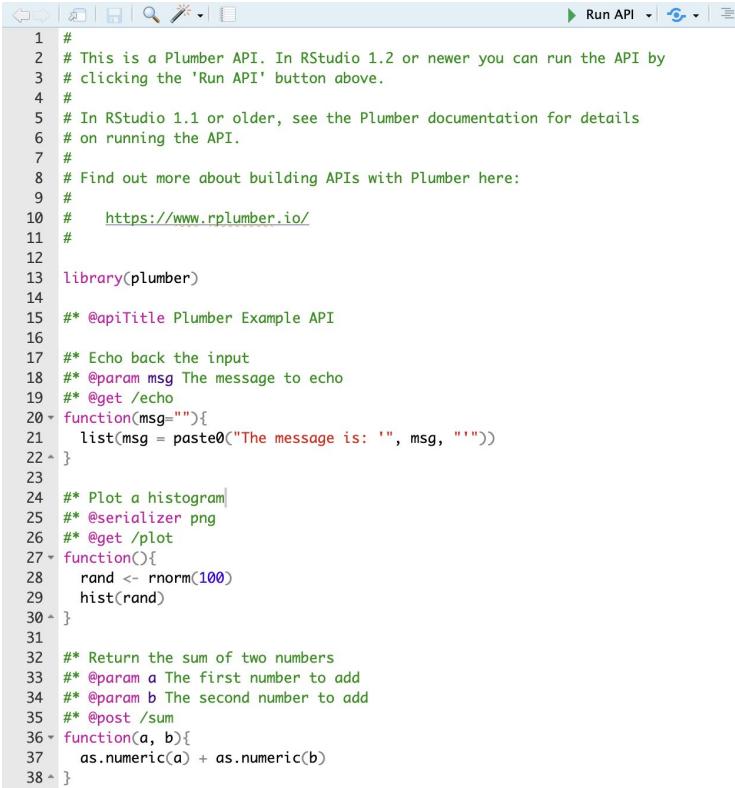


**POST**

## plumber/plumber.R Basic plumber API implementations



### Simple plumber.R



A screenshot of the RStudio IDE showing the contents of a file named "plumber.R". The code defines a Plumber API with two endpoints: one for echoing input and another for plotting a histogram. It also includes a function for summing two numbers.

```
1 #  
2 # This is a Plumber API. In RStudio 1.2 or newer you can run the API by  
3 # clicking the 'Run API' button above.  
4 #  
5 # In RStudio 1.1 or older, see the Plumber documentation for details  
6 # on running the API.  
7 #  
8 # Find out more about building APIs with Plumber here:  
9 #  
10 #   https://www.rplumber.io/  
11 #  
12  
13 library(plumber)  
14  
15 #* @apiTitle Plumber Example API  
16  
17 #* Echo back the input  
18 #* @param msg The message to echo  
19 #* @get /echo  
20 ~ function(msg=""){  
21   list(msg = paste0("The message is: '", msg, "'"))  
22 ~ }  
23  
24 #* Plot a histogram  
25 #* @serializer png  
26 #* @get /plot  
27 ~ function(){  
28   rand <- rnorm(100)  
29   hist(rand)  
30 ~ }  
31  
32 #* Return the sum of two numbers  
33 #* @param a The first number to add  
34 #* @param b The second number to add  
35 #* @post /sum  
36 ~ function(a, b){  
37   as.numeric(a) + as.numeric(b)  
38 ~ }
```

### Advantages

- Great starting point
- Numerous examples and clear documentation
- Initial start template supported in RStudio IDE

### Disadvantages

- Limited testing solutions available
- Default options, such as serialization, might be challenging to understand

### Best-use scenarios

- Low complexity applications
- Building proofs of concept APIs
- For users inexperienced with Plumber

POST

## plumber/pr Modifying pr object



plumber.R or entrypoint.R

Modify plumber object in plumber.R

```
## @plumber
function(pr) {
  pr |>
    pr_set_api_spec(
      function(spec) {
        spec$paths[["/echo"]]$get$summary <- "Custom /echo title"
        spec # Return the updated specification
      }
    )
}
```

Modify plumber object in entrypoint.R

```
# entrypoint.R
library(plumber)

# Load the API
pr <- plumb("plumber.R") |>
  pr_set_api_spec(
    function(spec) {
      spec$paths[["/plot"]]$get$summary <- "Plot endpoint custom title"
      spec # Return the updated specification
    }
  )

# Must return a Plumber object when using `entrypoint.R`
pr
```

POST

**plumber/R6** Plumber is R6 Class



## plumber and R6

```
> pr <- plumb("plumber.R") ←  
> pr  
# Plumber router with 3 endpoints, 4 filters, and 0 sub-routers.  
# Use `pr_run()` on this object to start the API.  
└─[queryString]  
└─[body]  
└─[cookieParser]  
└─[sharedSecret]  
└─/echo (GET)  
└─/plot (GET)  
└─/sum (POST)  
> class(pr)  
[1] "Plumber" "Hookable" "R6"
```

A screenshot of an RStudio interface showing the code for "plumber.R". The code defines a Plumber router with three endpoints: /echo (GET), /plot (GET), and /sum (POST). It includes echo, plot, and sum filters. The /echo filter echoes the input message. The /plot filter plots a histogram of random data. The /sum filter adds two numbers. The code also includes documentation and links to the Plumber API and RStudio help.

```
1 # This is a Plumber API. In RStudio 1.2 or newer you can run the API by  
2 # clicking the 'Run API' button above.  
3 # In RStudio 1.1 or older, see the Plumber documentation for details  
4 # on running the API.  
5 # Find out more about building APIs with Plumber here:  
6 # https://www.rplumber.io/  
7 #  
8 library(plumber)  
9 #* @apiTitle Plumber Example API  
10 #* Echo back the input  
11 #* @param msg The message to echo  
12 #* @get /echo  
13 function(msg){  
14   list(msg = paste0("The message is: ", msg, ""))  
15 }  
16 #* Plot a histogram  
17 #* @serializer png  
18 #* @get /plot  
19 #* @function()  
20 rand <- rnorm(100)  
21 hist(rand)  
22 }  
23 #* Return the sum of two numbers  
24 #* @param a The first number to add  
25 #* @param b The second number to add  
26 #* @post /sum  
27 #* @function(a, b){  
28   as.numeric(a) + as.numeric(b)  
29 }  
30 }  
31 #*  
32 #*  
33 #*  
34 #*  
35 #*  
36 #*  
37 #*  
38 }
```



## Simple plumber.R + entrypoint.R

A screenshot of the RStudio IDE showing an R script named "entrypoint.R". The code defines a Plumber API endpoint for plotting. The RStudio interface includes tabs for "Source on Save", "Run", and "Source".

```
1 # entrypoint.R
2 library(plumber)
3
4 # Load the API
5 pr <- plumb("plumber.R") |>
6   pr_set_api_spec(
7     function(spec) {
8       spec$paths[["/plot"]]$get$summary <- "Plot endpoint custom title"
9       spec # Return the updated specification
10    }
11  )
12
13 # Must return a Plumber object when using `entrypoint.R`
14 pr
15
```

### Advantages

- Similar to global.R in Shiny
- Allows for more customization

### Disadvantages

- Limited testing solutions available
- Needs some familiarity with R6

### Best-use scenarios

- Low complexity applications
- Building proof of concept
- Extending knowledge about R6 class

POST

## plumber/code Plumber as a code



Wow! You can create a plumber without plumber.R

```
pr <- pr() |>
  pr_handle(
    methods = "GET",
    path = "/echo",
    handler = function(msg "") {
      list(msg = paste0("The message is: ", msg, ""))
    },
    comments = "Echo back the input",
    description = "Returns message back to the user.",
    params = list(
      "msg" = list(
        type = "string",
        required = TRUE,
        desc = "The message to echo"
      )
    ),
    tags = "echo",
    serializer = plumber::serializer_unboxed_json()
  )
```

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

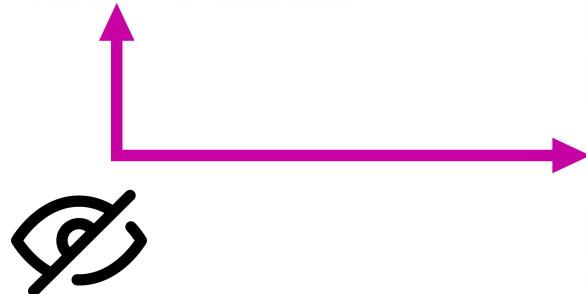
**POST**

**plumber/pr** Modifying pr object



Do whatever you want!

```
pr_set_api_spec(  
  function(spec) {  
    spec$paths[["/pet/findByTags"]] <- NULL  
    spec # Return the updated specification  
  })
```



Create hidden endpoint!

**pet** Everything about your Pets Find out more: <http://swagger.io> ✓

<b>POST</b>	/pet	Add a new pet to the store	
<b>PUT</b>	/pet	Update an existing pet	
<b>GET</b>	/pet/findByStatus	Finds Pets by status	
<b>GET</b>	/pet/findByTags	Finds Pets by tags	
<b>GET</b>	/pet/{petId}	Find pet by ID	
<b>POST</b>	/pet/{petId}	Updates a pet in the store with form data	
<b>DELETE</b>	/pet/{petId}	Deletes a pet	
<b>POST</b>	/pet/{petId}/uploadImage	uploads an image	

**POST**

## **plumber/pr** Modifying pr object



Do whatever you want!

```
pr_set_serializer()  
pr_set_docs_callback()  
pr_set_api_spec()  
pr_handle()  
pr_mount()  
pr_hooks()  
...
```

- Use functional paradigm

```
pr$setParsers()  
pr$setErrorHandler()  
pr$set404Handler()  
pr$setApiSpec()  
pr$mount()  
pr$registerHook()
```

- Function defined in R6 plumber object

- Build your own custom solution

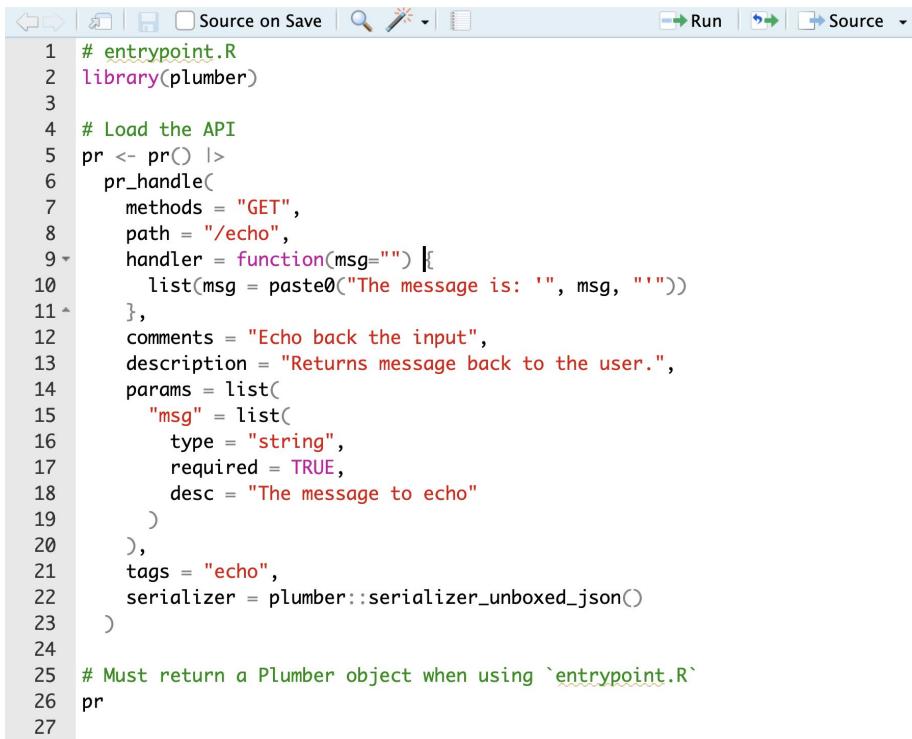
```
# Advance extension of plumber possibilities  
plumber:::plumber$set("active", "special_field", special_field_function)  
  
# Or shared functionalities used across multiple APIs  
extend_plumber(plumber:::plumber)
```

**POST**

**plumber/entrypoint.R** Build API only with entrypoint.R



Wow! You can make everything in entrypoint.R



The image shows a screenshot of the RStudio IDE. The top menu bar includes "File", "Edit", "Source", "Run", and "Help". Below the menu is a toolbar with icons for back, forward, search, and file operations. The main workspace shows the R code for "entrypoint.R". The code defines a Plumber API route for "GET /echo" that returns the input message. It includes comments, descriptions, parameters, and a serializer. The code ends with a note about returning a Plumber object.

```
1 # entrypoint.R
2 library(plumber)
3
4 # Load the API
5 pr <- pr() |>
6   pr_handle(
7     methods = "GET",
8     path = "/echo",
9     handler = function(msg="") {
10       list(msg = paste0("The message is: ", msg, ""))
11     },
12     comments = "Echo back the input",
13     description = "Returns message back to the user.",
14     params = list(
15       "msg" = list(
16         type = "string",
17         required = TRUE,
18         desc = "The message to echo"
19       )
20     ),
21     tags = "echo",
22     serializer = plumber::serializer_unboxed_json()
23   )
24
25 # Must return a Plumber object when using `entrypoint.R`
26 pr
```

## Advantages

- Build or modify Plumber APIs using R code
- Ability to generalize parts of your code
- Convenience with default parameters

## Disadvantages

- Most documentation focuses on Plumber.R
- Requires additional research on Plumber

## Best-use scenarios

- Greater control over Plumber routes
- Extract components for use in multiple APIs
- Managing increasing app complexity
- Defining custom filter functionalities



Wow! Your plumber API can be a package.

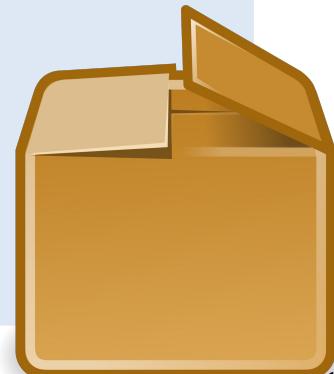
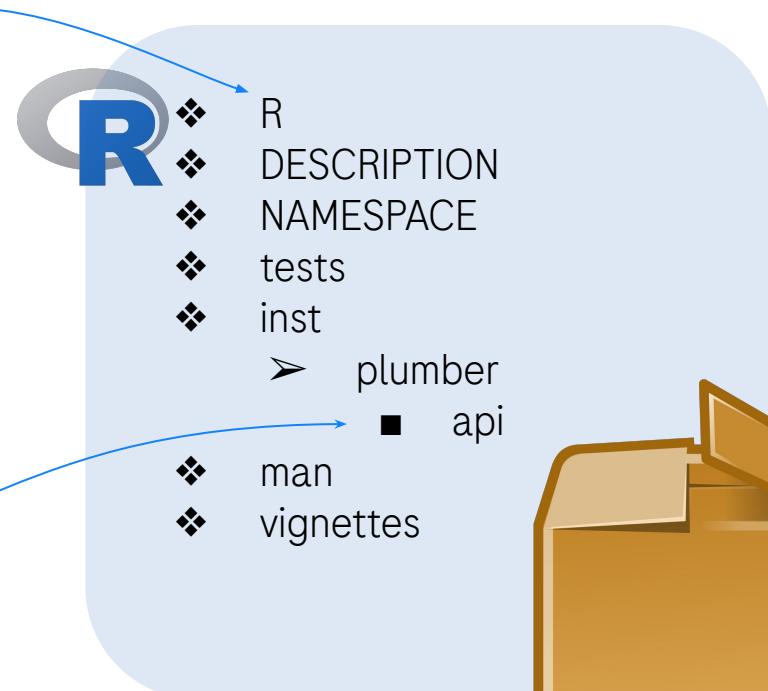
```
'#' Hello world
'#'
#' @return single character vector
#' @export
#'
#' @examples
#' echo("abc")
echo <- function(msg = "", req, res) {
  list(msg = paste0("The message is: '", msg, "'"))
}

#
# This is a Plumber API. You can run the API by clicking
# the 'Run API' button above.
#
# Find out more about building APIs with Plumber here:
#
#   https://www.rplumber.io/
#
library(plumber)

## @apiTitle Plumber Example API
## @apiDescription Plumber example description.

## Echo back the input
## @param msg The message to echo
## @get /echo
echo

## Hello message
## @get /hello
hello
```



POST

**plumber/devtools** Use R community devtools!

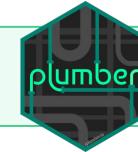


Wow! Use nice R package development workflow.



**POST**

## **plumber/package** Dependency management



Wow! Better dependency management with DESCRIPTION file.

**Package:** Template4

**Type:** Package

**Title:** What the Package Does (Title Case)

**Version:** 0.1.0

**Author:** Who wrote it

**Maintainer:** The package maintainer <yourself@somewhere.net>

**Description:** More about what it does (maybe more than one line)

    Use four spaces when indenting paragraphs within the Description.

**Packages:**

    plumber

**License:** What license is it under?

**Encoding:** UTF-8

**LazyData:** true

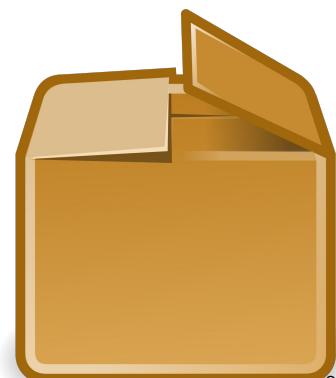
**Suggests:**

    testthat (>= 3.0.0)

**Config/testthat/edition:** 3

**Roxygen:** list(markdown = TRUE)

**RoxygenNote:** 7.3.1



**POST**

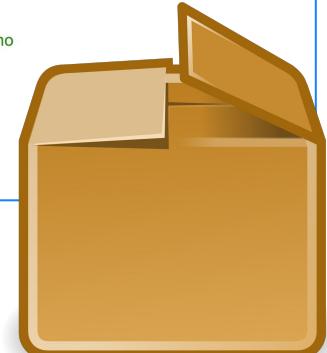
## plumber/package Multiple API per package



Wow! Plumber can discover API in your package.

```
> devtools::load_all()  
i Loading Template4  
> plumber::available_apis("Template4")  
Available Plumber APIs:  
* Template4  
  - api ←  
> plumber::available_apis("plumber")  
Available Plumber APIs:  
* plumber  
  - 01-append  
  - 02-filters  
  - 03-github  
  - 04-mean-sum  
  - 05-static  
  - 06-sessions  
  - 07-mailgun  
  - 08-identity  
  - 09-content-type  
  - 10-welcome  
  - 11-car-inventory  
  - 12-entrypoint  
  - 13-promises  
  - 14-future  
  - 15-openapi-spec  
  - 16-attachment  
  - 17-arguments
```

```
#  
# This is a Plumber API. You can run the API by clicking  
# the 'Run API' button above.  
#  
# Find out more about building APIs with Plumber here:  
#  
#   https://www.rplumber.io/  
  
library(plumber)  
  
## @apiTitle Plumber Example API  
## @apiDescription Plumber example description.  
  
## Echo back the input  
## @param msg The message to echo  
## @get /echo  
echo  
  
## Hello message  
## @get /hello  
hello
```





Wow! You can zip your plumber!



### Advantages

- Utilize the R community devtools
- Simplified testing with testthat
- Improved dependency management
- Easy CI-CD integration with existing components

### Disadvantages

- Requires more experience with R
- Can be more time-consuming compared to a simple Plumber script

### Best-use scenarios

- Everywhere!

POST

## plumber/package/code Combine package and code



Combine plumber as code and as package.

```
#' Run plumber object
#'
#' @param ... All parameters for run API.
#' @param pr Plumber object by default prepare_api()
#' @export
#'
#' @examples
#' run_api()
run_api <- function(..., pr = prepare_api()) {
  pr$run(...)
}
```

Possibility to extend run functionality:

- Multi environment support
- Apply custom decorator rules

```
#' Create plumber object
#'
#' @return plumber object
#' @export
#'
#' @examples
#' pr <- prepare_api()
prepare_api <- function() {
  # The package needs to be installed for multisession.
  future::plan(future::sequential())

  plumber::pr() |>
    plumber::pr_handle(
      methods = "GET",
      path = "/echo",
      handler = echo,
      comments = "Echo back the input",
      description = "Returns message back to the user.",
      params = list(
        "msg" = list(
          type = "string",
          required = TRUE,
          desc = "The message to echo"
        )
      ),
      tags = "hello_world",
      serializer = plumber::serializer_unboxed_json()
    ) |>
    plumber::pr_handle(
      methods = "GET",
      path = "/hello",
      handler = promise_decorate(hello),
      comments = "Say hello!",
      description = "Your first function.",
      tags = "hello_world",
      serializer = plumber::serializer_unboxed_json()
    ) |>
    plumber::pr_handle(
      methods = "GET",
      path = "/slow",
      handler = promise_decorate(slow),
      comments = "Slow endpoint!",
      tags = "hello_world",
      serializer = plumber::serializer_unboxed_json()
    )
}
```



To minimize complexity, consider using a decorator.

```
plumber::pr_handle(  
  methods = "GET",  
  path = "/slow",  
  handler = promise_decorate(slow),  
  comments = "Slow endpoint!",  
  tags = "hello_world",  
  serializer = plumber::serializer_unboxed_json()  
)
```

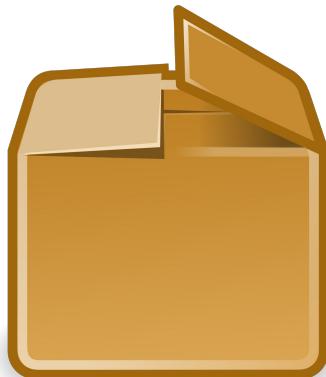
```
'# Decorate function with future_promise  
' # @param fun Function to decorate  
' # @return The input function decorated with future_promise  
promise_decorate <- function(fun) {  
  # Define  
  if (isFALSEgetOption("plumber.use.promises", TRUE))) return(fun)  
  
  # Create new function with future_promise evaluation  
  rlang::new_function(alist(... = ), rlang::expr({  
    promises::future_promise({  
      fun(...)  
    })  
  }), env = rlang::current_env())  
}
```



Wow! Your API as a code with all R package benefits.

```
#' Create plumber object
#'
#' @return plumber object
#' @export
#'
#' @examples
#' pr <- prepare_api()
# prepare_api <- function() {
#   # The package needs to be installed for multisession.
future::plan(future::sequential())
#}

plumber::pr() |>
  plumber::pr_handle(
    methods = "GET",
    path = "/echo",
    handler = echo,
    comments = "Echo back the input",
    description = "Returns message back to the user.",
    params = list(
      "msg" = list(
        type = "string",
        required = TRUE,
        desc = "The message to echo"
      )
    ),
    tags = "hello_world",
    serializer = plumber::serializer_unboxed_json()
) |>
  plumber::pr_handle(
    methods = "GET",
    path = "/hello",
    handler = promise_decorate(hello),
    comments = "Say hello!",
    description = "Your first function.",
    tags = "hello_world",
    serializer = plumber::serializer_unboxed_json()
) |>
  plumber::pr_handle(
    methods = "GET",
    path = "/slow",
    handler = promise_decorate(slow),
    comments = "Slow endpoint!",
    tags = "hello_world",
    serializer = plumber::serializer_unboxed_json()
)
}
```



## Advantages

- Includes all benefits of the package approach
- Excellent for validation

## Disadvantage

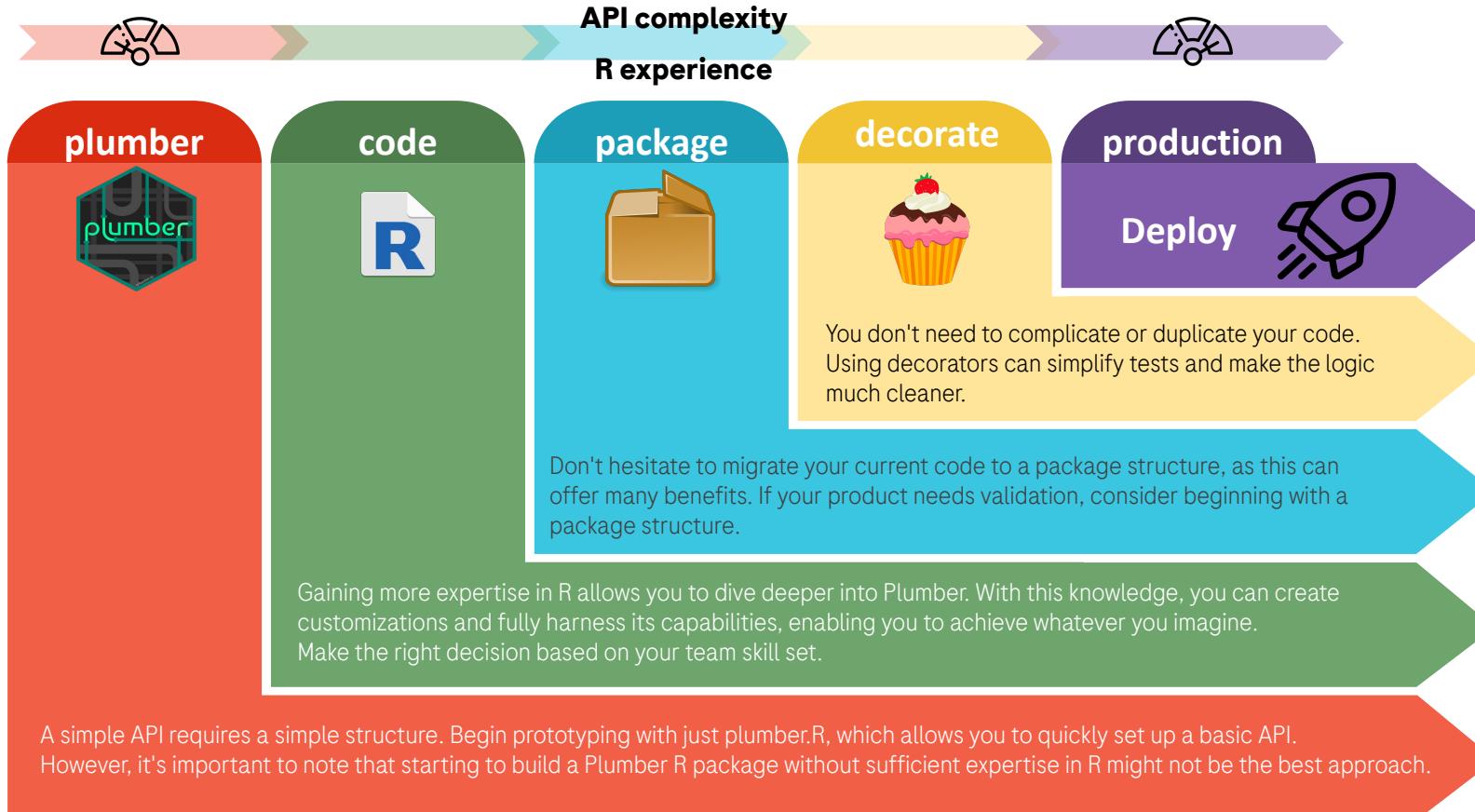
- Requires more experience with R
- Can be more time-consuming compared to a simple plumber script

## Best-use scenarios

- Everywhere!
- Universal framework

PUT

## /select How to choose the right approach?





- Visit the GitHub repository at <https://github.com/r-world-devs/useR2024-mastering-plumber-api> to check the templates.



- See our poster and ask us questions about our internal MINT+ application.
- Explore our business use case in the presentation titled "MINT+: Web App with R Brains for SDTM Automation" from the virtual useR 2024 conference.



- Get familiar with R6.
- Read plumber code.



**DELETE**

**/presentation** Thank you for your attention!

**Doing now what patients need next**