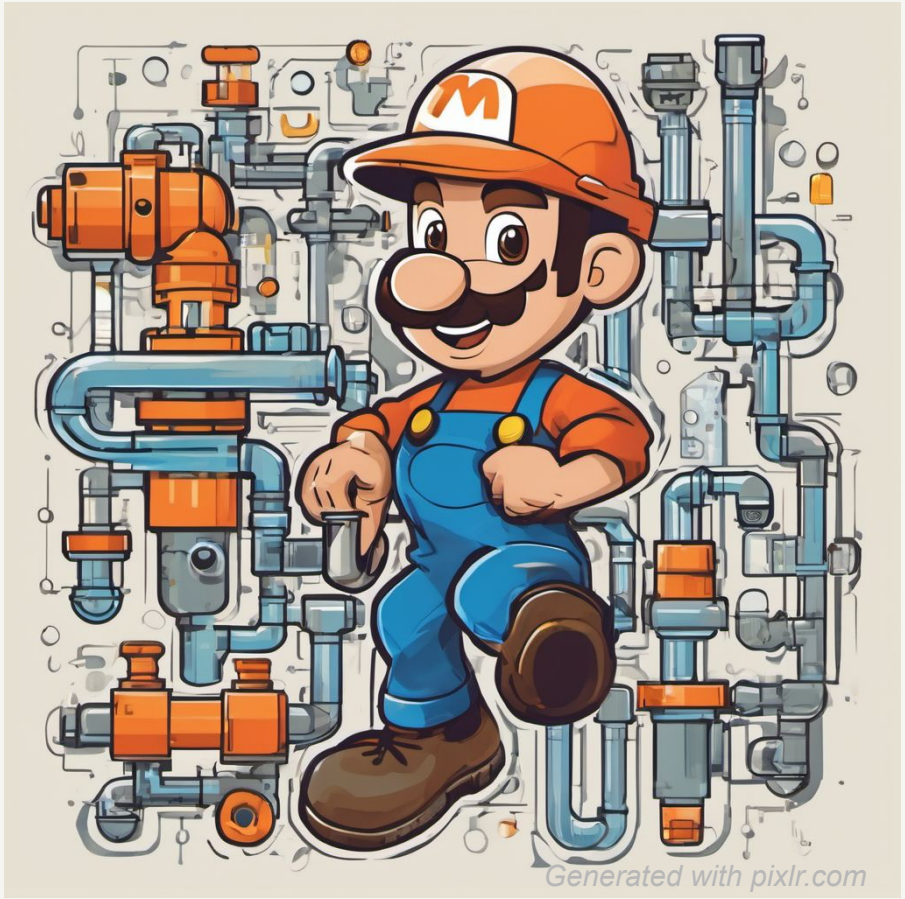


Mastering Plumber Structure: Your API's Solutions

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

10 July 2024

Plumber API

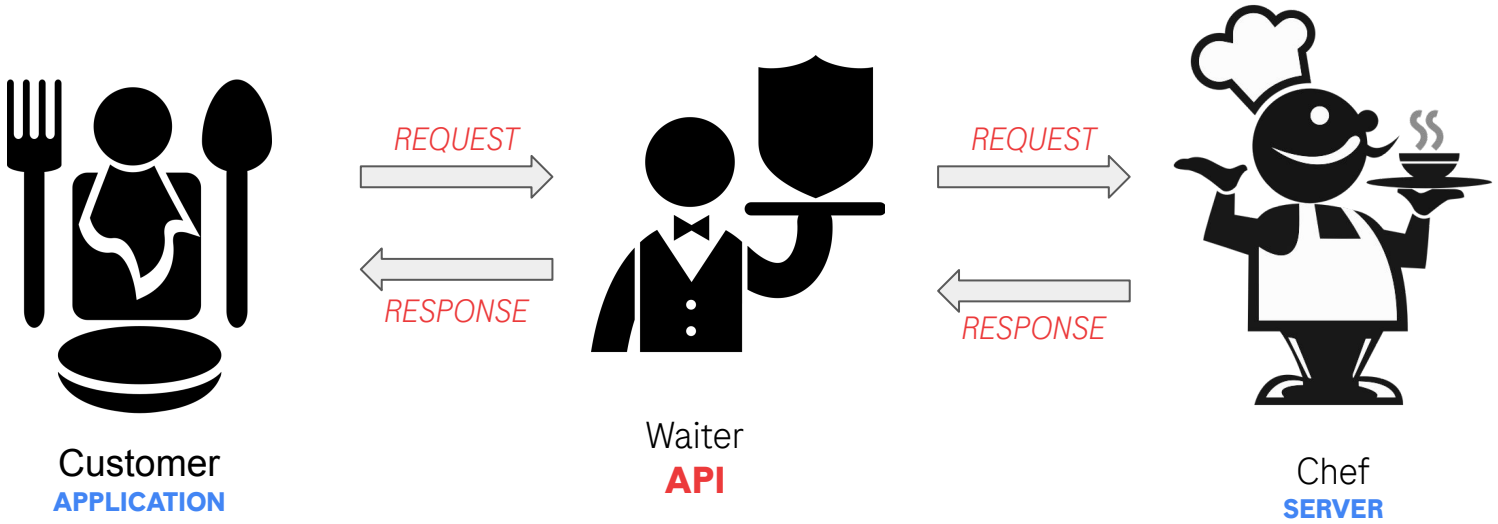
Table of contents

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

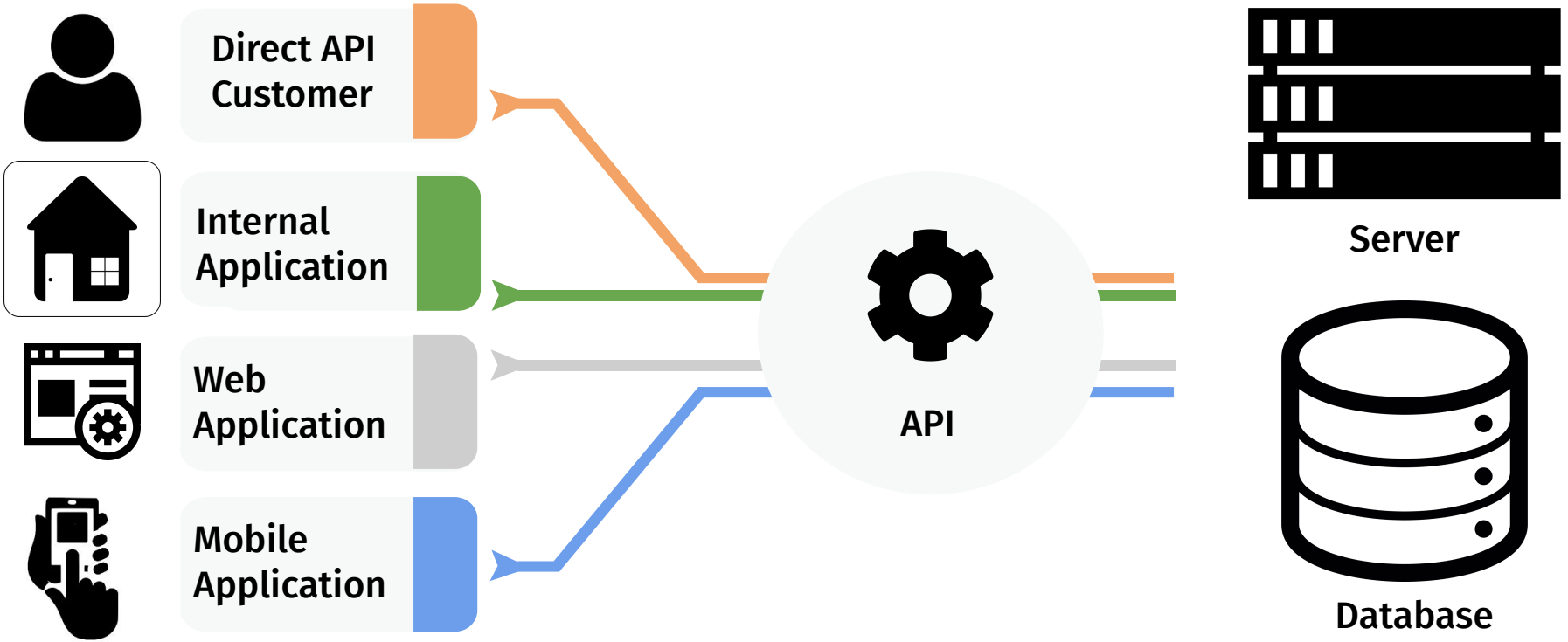
What is an API?

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 API

Grammarly API

PayPal

Spotify

Dropbox

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.

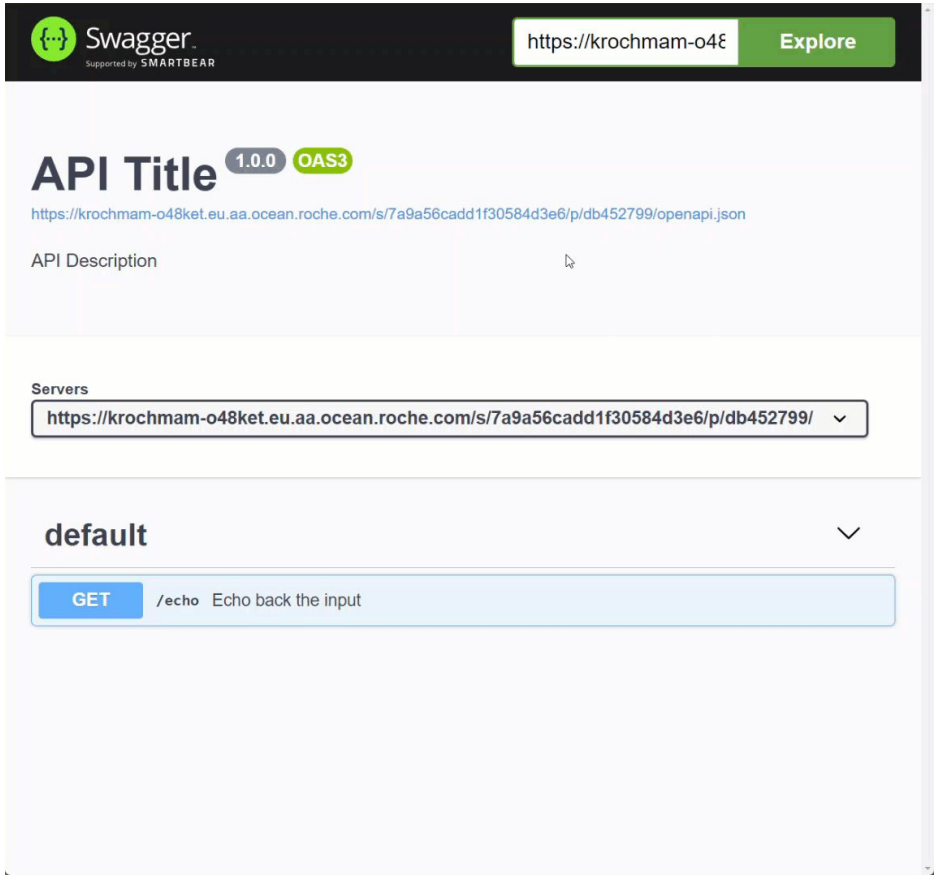
plumber.R

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

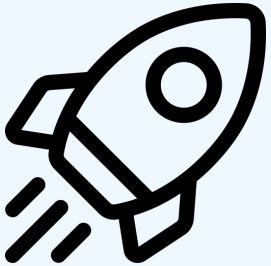
QUICKSTART

```
R 4.3.3 · ~/
> 
> 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



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



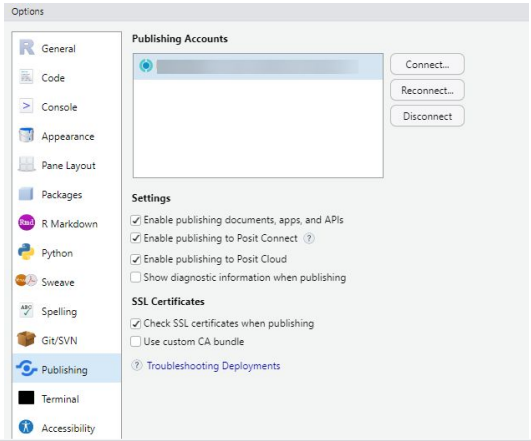
DEPLOYMENT / HOSTING

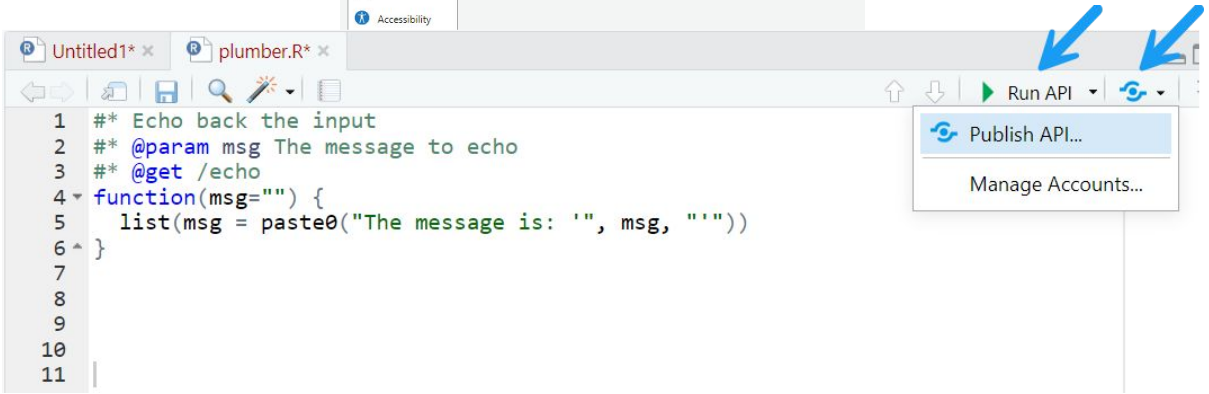
posit™

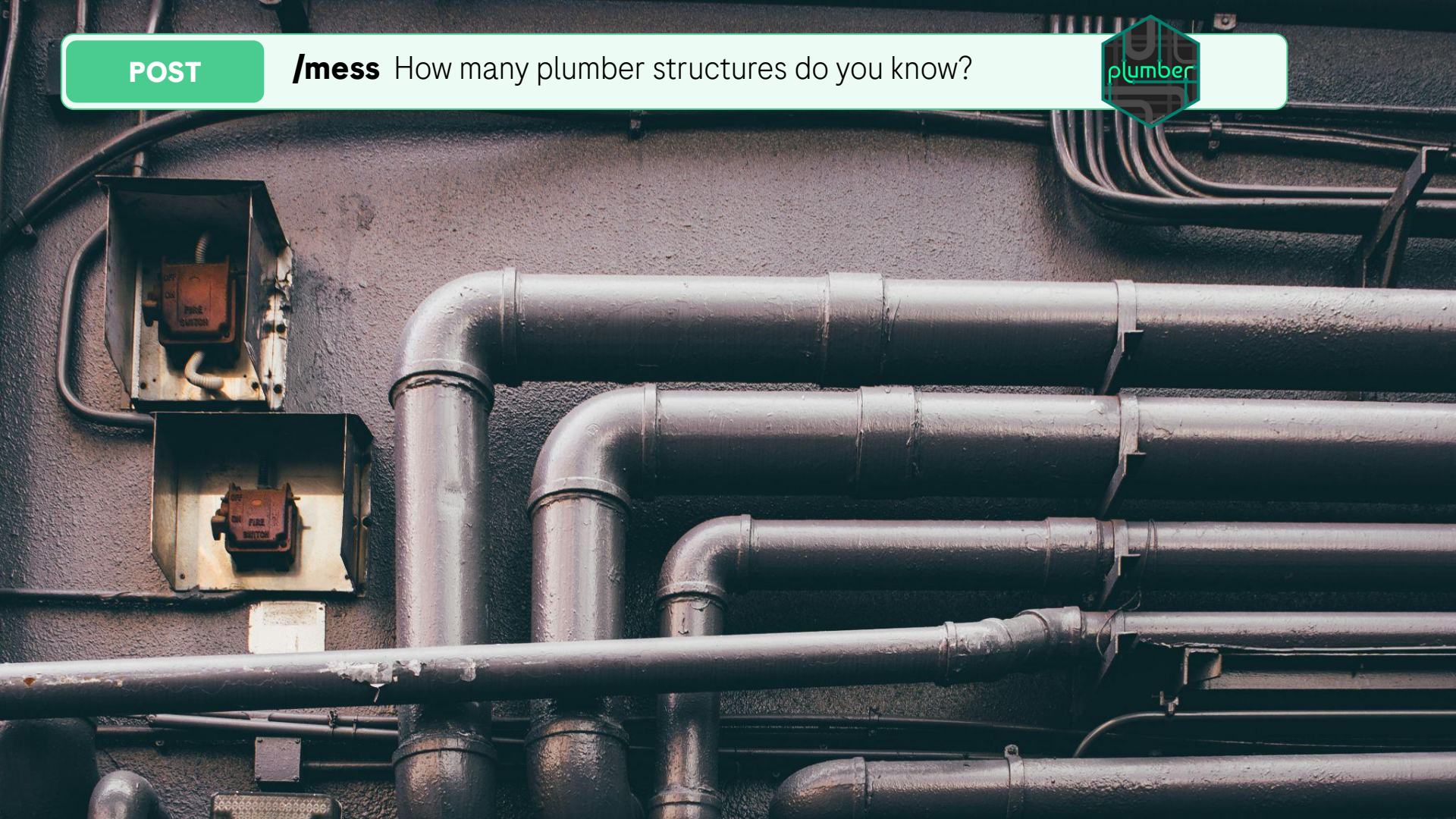
DigitalOcean

Containers as a Service (CaaS)

Docker






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

POST **plumber/plumber.R** Basic plumber API implementations

Roche

useR! Salzburg 2024

Simple plumber.R

Run API

```
#
# This is a Plumber API. In RStudio 1.2 or newer you can run the API by
# clicking the 'Run API' button above.
#
# In RStudio 1.1 or older, see the Plumber documentation for details
# on running the API.
#
# Find out more about building APIs with Plumber here:
#
#    https://www.rplumber.io/
#

library(plumber)

#* @apiTitle Plumber Example API

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

#* Plot a histogram
#* @serializer png
#* @get /plot
function(){
  rand <- rnorm(100)
  hist(rand)
}

#* Return the sum of two numbers
#* @param a The first number to add
#* @param b The second number to add
#* @post /sum
function(a, b){
  as.numeric(a) + as.numeric(b)
}
```

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

Roche

useR! Salzburg 2024

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

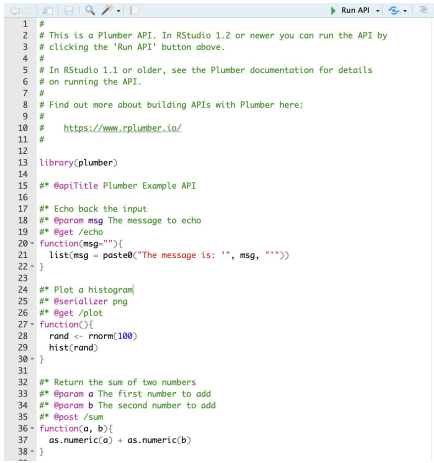
```
# entrypoint.R
library(plumber)

# 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"
```

plumber.R



```
#
# This is a Plumber API. In RStudio 1.2 or newer you can run the API by
# clicking the 'Run API' button above.
#
# In RStudio 1.1 or older, see the Plumber documentation for details
# on running the API.
#
# Find out more about building APIs with Plumber here:
#
#    https://www.rplumber.io/
#

library(plumber)

#* @apiTitle Plumber Example API

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

#* Plot a histogram
#* @serializer png
#* @get /plot
function(){
  rand <- rnorm(100)
  hist(rand)
}

#* Return the sum of two numbers
#* @param a The first number to add
#* @param b The second number to add
#* @post /sum
function(a, b){
  as.numeric(a) + as.numeric(b)
}
```

POST **plumber/entrypoint.R** Entrypoint and plumber files

Roche

useR! Salzburg 2024

Simple plumber.R + entrypoint.R

Source on Save | Run | 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

Roche

useR! Salzburg 2024

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()
  )
```

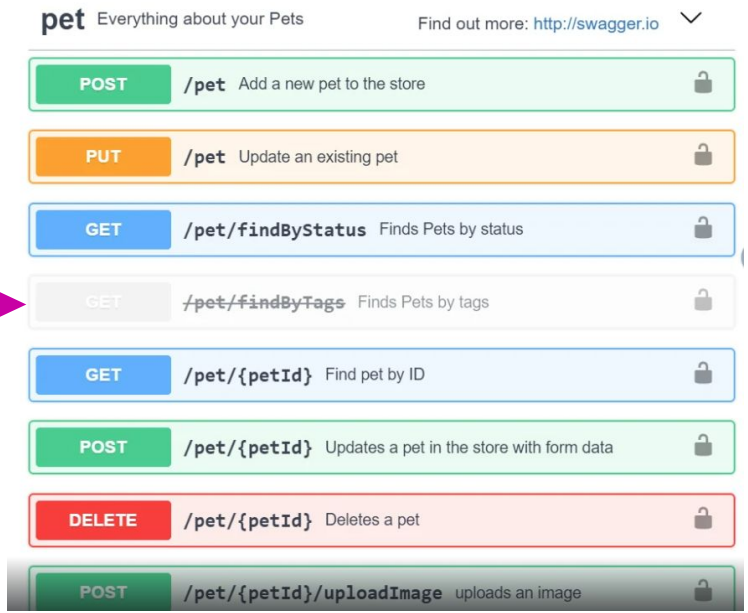
```
#* Echo back the input
#* @param msg The message to echo
#* @get /echo
function(msg=""){
  list(msg = paste0("The message is: '", msg, "'"))
}
```

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!



POST **plumber/pr** Modifying pr object

Roche

useR! Salzburg 2024

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

Roche

useR! Salzburg 2024

Wow! You can make everything in entrypoint.R

Source on Save | Run | Source

```r
# entrypoint.R
library(plumber)

# Load the API
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()
  )

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

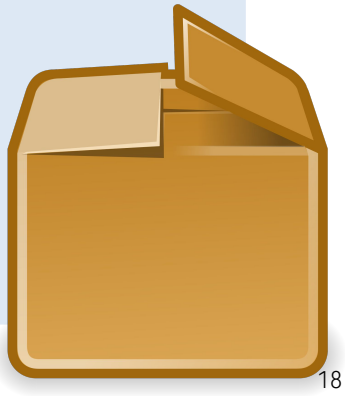
POST **plumber/package** Plumber as a package

Wow! Your plumber API can be a package.

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

```r
#
# 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
```

- R
- DESCRIPTION
- NAMESPACE
- tests
- inst
  - plumber
    - api
- man
- vignettes



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

Roche

useR! Salzburg 2024

Wow! Use nice R package development workflow.

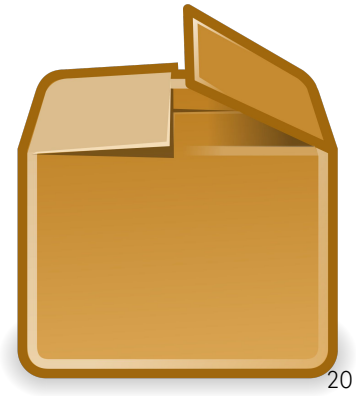
POST **plumber/package** Dependency management

Roche

useR! Salzburg 2024

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

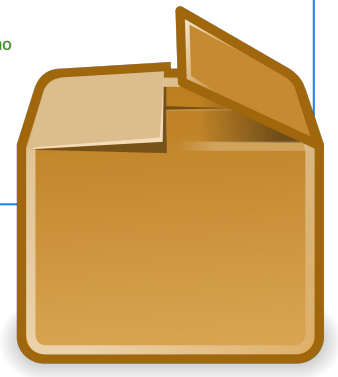
Roche

useR! Salzburg 2024

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
```



POST **plumber/package** Plumber as a package

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

Roche

useR! Salzburg 2024

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(...)
}
```

```
#' 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()
    )
}
```

Possibility to extend run functionality:

- Multi environment support
- Apply custom decorator rules

POST **plumber/decorate** Decorate functions

Roche

useR! Salzburg 2024

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 (isFALSE(getOption("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())

}
```

POST **plumber/package/code** Plumber as a package

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?

Roche

useR! Salzburg 2024

API complexity

R experience

plumber

code

package

decorate

production

Deploy

You don't need to complicate or duplicate your code. Using decorators can simplify tests and make the logic much cleaner.

Don't hesitate to migrate your current code to a package structure, as this can offer many benefits. If your product needs validation, consider beginning with a package structure.

Gaining more expertise in R allows you to dive deeper into Plumber. With this knowledge, you can create customizations and fully harness its capabilities, enabling you to achieve whatever you imagine.
Make the right decision based on your team skill set.

A simple API requires a simple structure. Begin prototyping with just plumber.R, which allows you to quickly set up a basic API.
However, it's important to note that starting to build a Plumber R package without sufficient expertise in R might not be the best approach.

PUT **/learn_more** Learn more

- 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