There is a small examples of how to establish monitoring of Nginx worker’s load. In a few words, it is possible to set up an internal server that would sit on an arbitrary Nginx worker process and accumulate various data from all the workers. This data could be retrieved via specified interface configured in the Nginx configuration script. In the example, the internal server collects the number of requests and bytes that were sent back to clients from each worker process. This data is accessible in *JSON* format via a dedicated virtual server listening on port *8020*. Say, to retrieve the current load, we can simply use *curl* and *jq* (to pretty-print JSON).

curl '<http://127.0.0.1:8020/>' | jq  
[  
 "2019-04-22T14:29:04Z",  
 {  
 "5910": [  
 "2019-04-22T14:31:34Z",  
 {  
 "bytesSent": 17751,  
 "requests": 97,  
 "meanBytesSent": 183  
 }  
 ],  
 "5911": [  
 "2019-04-22T14:31:31Z",  
 {  
 "bytesSent": 549,  
 "requests": 3,  
 "meanBytesSent": 183  
 }  
 ]  
 }  
]

Now I want to show how to retrieve this data repeatedly and render it on an interactive dashboard online using R and a *Shiny* application.

Below is the annotated code saved in a file *load\_monitoring.r*.

|  |  |
| --- | --- |
| 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 | library(shiny) library(plotly) library(jsonlite)  ui <- fluidPage(  fluidRow(  headerPanel(h1("Nginx workers load arbitration", align = "center"),  "Nginx workers load arbitration")),  fluidRow(  wellPanel(div(align = "center",  div(style = "display: inline-block; margin-right: 20px",  textInput("i\_url", NULL, "<http://127.0.0.1:8020/>",  width = "200px")), span(),  div(style = "display: inline-block; margin-right: 20px",  radioButtons("rb\_mode", NULL,  c("Requests" = "reqs",  "Bytes\_sent" = "bsent"),  selected = "reqs", inline = TRUE)),  div(style = "display: inline-block",  actionButton("b\_reset", "Reset Traces"))))),  fluidRow(  div(plotlyOutput("plot"), id = "graph")) )  server <- function(input, output, session) {  values <- reactiveValues()  values$init <- TRUE  values$load <- list("", list())  values$pids\_prev <- list()   observe({  invalidateLater(5000, session)   if (class(values$load) != "try-error") {  values$pids\_prev <- names(values$load[[2]])  }  values$load <- try(fromJSON(input$i\_url))   m <- `if`(input$rb\_mode == "reqs", 2, 1)   if (class(values$load) == "try-error" ||  length(values$load[[2]]) == 0) {  invalidateLater(1000, session)  } else {  pids <- names(values$load[[2]])   if (values$init) {  values$init <- FALSE  values$p <- plot\_ly(type = "scatter",  mode = "lines",  colors = "YlOrRd")   for (i in 1:length(values$load[[2]])) {  xs <- as.POSIXct(values$load[[2]][[i]][[1]],  format = "%Y-%m-%dT%H:%M:%S")  values$p <- add\_trace(values$p,  name = paste(pids[i],  names(values$load[[2]][[i]][[2]][m])),  x = xs,  y = values$load[[2]][[i]][[2]][[m]],  line = list(width = 2)) %>%  add\_annotations(  x = xs,  y = values$load[[2]][[i]][[2]][[m]],  text = "",  showarrow = TRUE, arrowcolor = "#bbb")  }  values$p <- layout(values$p, yaxis = list(range = 0))   output$plot <- renderPlotly(values$p)  } else {  vs <- list()  xs <- list()  ts <- list()  len <- length(pids)   if (length(names(values$load[[2]])) !=  length(values$pids\_prev) ||  length(setdiff(names(values$load[[2]]),  values$pids\_prev)) > 0) {  invalidateLater(1000, session)  values$init <- TRUE  } else {  i <- 1  while (i <= len) {  vs[[i]] <- list(values$load[[2]][[i]][[2]][[m]])  xs[[i]] <- list(values$load[[2]][[i]][[1]])  ts[[i]] <- i  i <- i + 1  }   plotlyProxy("plot", session) %>%  plotlyProxyInvoke("extendTraces",  list(x = xs, y = vs), ts)  }  }  }  }  )   observeEvent(input$b\_reset, {  values$init <- TRUE  }  )   observeEvent(input$rb\_mode, {  values$init <- TRUE  }  ) } |

In lines *1–3* all required libraries are loaded: *shiny* for UI, *plotly* for interactive plotting, and *jsonlite* for reading JSON data. The user interface is built in lines *5–23*. It consists of the header (lines *6–8*), a panel with control widgets (lines *9–20*), and the plot area (lines *21–22*).

In line *25–110* a Shiny server that would render data online, is defined. In lines *26–29* a number of *reactive values* are declared and initialized: they will be used in reactive observer *observe* defined in lines *31–99*. The observer runs every *5* seconds (line *32*) retrieving new data from Nginx (line *37*) and plotting it on the dashboard in case of success (lines *44–97*). If retrieval fails then *observe* re-runs in *1* second (lines *41–43*).

There are two branches of execution on successful data retrieval: initialization of the plot (lines *47–70*), and extending traces (lines *71–97*). The initialization triggers when the reactive value of *values$load* is *TRUE*: in this phase the *plotly* object *values$p* gets initialized in calls to *plot\_ly*, *add\_trace*, and *add\_annotations*. The plot type is set to *scatter*, its mode is *lines*, and the color palette for traces is *YlOrRd* (lines *49–51*). The traces and annotations are added in a loop (lines *53–67*), the number of iterations depends on JSON data saved in the value *values$load* and corresponds to the number of Nginx worker processes. The annotations are empty string (they have value ): they are only needed for drawing arrows at the beginnings of the traces.

Extending traces comes after the initialization phase, and only if the *PIDs* of the Nginx worker processes (found in line *45*) did not change after the previous data retrieval (it gets checked in lines *77–82*). If the PIDs have changed then the plot will be redrawn in *1* second (lines *81–82*). A more graceful solution would be adding new traces dynamically, without redrawing of the whole plot, however this would be more challenging for such a simple example, and not very useful taking into account that workers’ PIDs shouldn’t change often in the normal case. So, when the PIDs do not change, the traces get extended with new values (lines *83–95*).

In lines *101–109* actions for the Reset button and the Mode radio-button are defined: they simply reset the value of *values$init* to *TRUE* in order to redraw the plot.

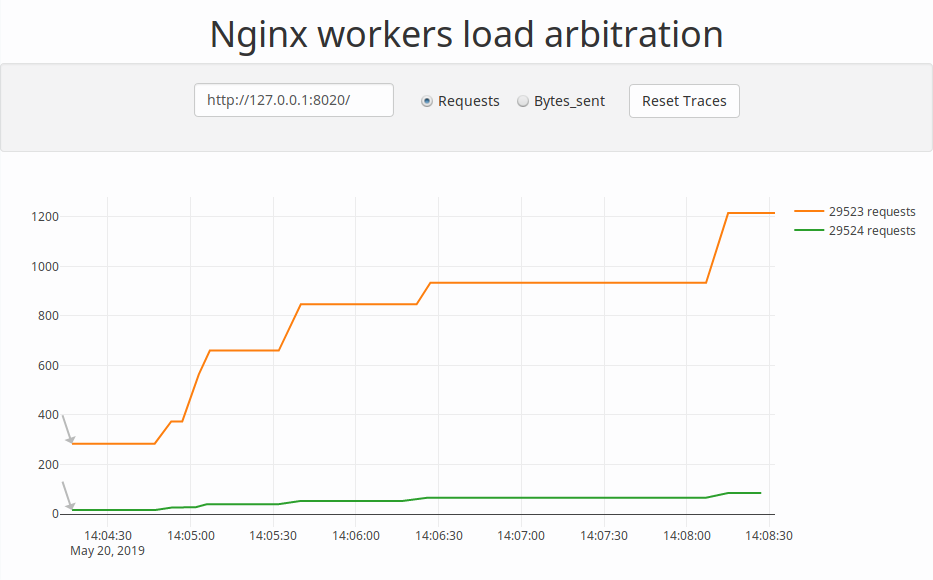
Now let’s start an *R shell* and run the server.

source("load\_monitoring.r")  
shinyApp(ui, server)  
  
Listening on <http://127.0.0.1:6678>

The application will open in a browser. Now run a number of requests to the Nginx server.

for i in {1..100} ; do curl '<http://127.0.0.1:8010/>' & done  
 ...  
for i in {1..100} ; do curl '<http://127.0.0.1:8010/>' & done  
 ...

The application in the bowser shall look like on the image below.

[](https://i2.wp.com/3.bp.blogspot.com/-mmaLUpKqpiM/XOLVLn36z4I/AAAAAAAABvs/N3WrXv_2nbIBR-zmzAhxAJACYJiBYURWwCLcBGAs/s1600/nwla3.png?ssl=1)

The application renders load of the Nginx workers in real-time. The address of the Nginx virtual stats server and types of traces can be altered using control widgets on the grayish panel above the plot.