{golem} has a function called run\_app(), which, in its original implementation, relies on calling runApp() on the two files contained in the inst/ folder of the package. But here comes the issue — **runApp() can’t be used with RStudio products (Shiny Server, Connect & Shinyapps.io)**, as it returns an error Can't call runApp() from within runApp(). The previous workaround was to use shinyApp(...) in the app.R file, but I like things to be stable so having various solutions didn’t seem like the best answer.

Two questions arise from that:

Why on earth a run\_app() function? Well, because it allows to deploy easily through command line. Using a run\_app() function allow more flexibility, if we succeed to design it to take parameters and to pass it to the app. For example, in Docker, we could use environment variables as function parameters. A flexible run\_app() function would also allow to change the behaviour of the app without having to tweak the app.R file we’re using for RStudio Products. Or, as you will see in this very blog post, a run\_app() function with parameters can also be used to change the behaviour of our local app 

Ok, so that was the story behind run\_app(). But why on earth this first implementation with runApp()? You’ll see in a minute why one would choose that function But, TL;DR, runApp is able to use local options defined into the function.

And these three functions (the one from the title) do not have the same behaviour, depending on where they are used:

* runApp() doesn’t work on RStudio products, but is the only way Docker and local calls can access options passed to the run\_app() function.
* shinyApp() and shinyAppDir() works likewise wherever you use them.
* RStudio production do not handle “local” options defined in the run\_app() function.
* There’s a fourth way, the one implemented in {golem}, which is fit for all scenario 

**Several ways to launch a Shiny App**

1. runApp() (old implementation of {golem}), which is a wrapper aroundshiny::runApp(system.file("app", package = "aaaaaa")).
2. shinyApp(), which is shiny::shinyApp(ui = app\_ui(), server = app\_server), the solution created by golem::add\_rstudioconnect\_file() and friends.
3. shinyAppDir(), which is shinyAppDir( system.file("app", package = "aaaaaa") ) — a necessary workaround for Shiny server if you wanted to call the app/ folder through the old implementation of {golem}.

Note: there is also shinyAppFile(), but its inner behaviour is the same as shinyAppDir(), so it’s not benchmarked here.

**One naive implementation**

**What we could have done there is simply leaving things open for the end user**, so that they have to chose the best implementation for their deployment use case. By doing something like:

run\_app <- function(

with = c("shinyApp", "runApp", "shinyAppDir")

) {

with <- match.arg(with)

if (with == "shinyApp"){

shiny::shinyApp(ui = app\_ui(), server = app\_server)

} else if (with == "runApp") {

shiny::runApp(system.file("app", package = "aaaa"))

} else if (with == "shinyAppDir") {

shiny::shinyAppDir(system.file("app", package = "aaaa"))

}

}

This might be the best answer as it leaves the choice to the user, but the question is still open: **what function should I use for my deployment**?

Let’s keep this function and use it for our benchmark.

**Side node**

If I refer to ?shinyApp:

You generally shouldn’t need to use these functions to create/run applications; they are intended for interoperability purposes. Read Below:

Create a Shiny app object

[shinyApp](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(ui = NULL, server = NULL, onStart = NULL,

options = [list](http://www.rdocumentation.org/packages/base/topics/list)(), uiPattern = "/", enableBookmarking = NULL)

[shinyAppDir](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(appDir, options = [list](http://www.rdocumentation.org/packages/base/topics/list)())

[shinyAppFile](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(appFile, options = [list](http://www.rdocumentation.org/packages/base/topics/list)())

[as.shiny.appobj](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(x)

# S3 method for shiny.appobj

[as.shiny.appobj](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(x)

# S3 method for list

[as.shiny.appobj](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(x)

# S3 method for character

[as.shiny.appobj](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(x)

[is.shiny.appobj](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)(x)

# S3 method for shiny.appobj

[print](http://www.rdocumentation.org/packages/base/topics/print)(x, ...)

# S3 method for shiny.appobj

[as.tags](http://www.rdocumentation.org/packages/htmltools/topics/as.tags)(x, ...)

Arguments

|  |  |
| --- | --- |
| **ui** | The UI definition of the app (for example, a call to fluidPage() with nested controls) |
| **server** | A server function |
| **onStart** | A function that will be called before the app is actually run. This is only needed for shinyAppObj, since in the shinyAppDir case, a global.R file can be used for this purpose. |
| **options** | Named options that should be passed to the runApp call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify width and height parameters which provide a hint to the embedding environment about the ideal height/width for the app. |
| **uiPattern** | A regular expression that will be applied to each GET request to determine whether the ui should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful. |
| **enableBookmarking** | Can be one of "url", "server", or "disable". This is equivalent to calling the enableBookmarking() function just before calling shinyApp(). With the default value (NULL), the app will respect the setting from any previous calls to enableBookmarking(). See enableBookmarking for more information. |
| **appDir** | Path to directory that contains a Shiny app (i.e. a server.R file and either ui.R or www/index.html) |
| **appFile** | Path to a .R file containing a Shiny application |
| **x** | Object to convert to a Shiny app. |
| **...** | Additional parameters to be passed to print. |

Value

An object that represents the app. Printing the object or passing it to [runApp](https://shiny.rstudio.com/reference/shiny/1.3.2/runApp.html) will run the app.

Description

These functions create Shiny app objects from either an explicit UI/server pair (shinyApp), or by passing the path of a directory that contains a Shiny app (shinyAppDir). You generally shouldn't need to use these functions to create/run applications; they are intended for interoperability purposes, such as embedding Shiny apps inside a knitr document.

Details

Normally when this function is used at the R console, the Shiny app object is automatically passed to the print() function, which runs the app. If this is called in the middle of a function, the value will not be passed to print() and the app will not be run. To make the app run, pass the app object to print() or runApp().

Examples

**## Only run this example in interactive R sessions**

**if (**[**interactive**](http://www.rdocumentation.org/packages/base/topics/interactive)**()) {**

[**options**](http://www.rdocumentation.org/packages/base/topics/options)**(device.ask.default = FALSE)**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(**

**ui =** [**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**numericInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/numericInput.html)**("n", "n", 1),**

[**plotOutput**](https://shiny.rstudio.com/reference/shiny/1.3.2/plotOutput.html)**("plot")**

**),**

**server = function(input, output) {**

**output$plot <-** [**renderPlot**](https://shiny.rstudio.com/reference/shiny/1.3.2/renderPlot.html)**(** [**plot**](http://www.rdocumentation.org/packages/graphics/topics/plot)**(**[**head**](http://www.rdocumentation.org/packages/utils/topics/head)**(cars, input$n)) )**

**}**

**)**

[**shinyAppDir**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(**[**system.file**](http://www.rdocumentation.org/packages/pkgload/topics/system.file)**("examples/01\_hello", package="shiny"))**

**# The object can be passed to runApp()**

**app <-** [**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(**

**ui =** [**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**numericInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/numericInput.html)**("n", "n", 1),**

[**plotOutput**](https://shiny.rstudio.com/reference/shiny/1.3.2/plotOutput.html)**("plot")**

**),**

**server = function(input, output) {**

**output$plot <-** [**renderPlot**](https://shiny.rstudio.com/reference/shiny/1.3.2/renderPlot.html)**(** [**plot**](http://www.rdocumentation.org/packages/graphics/topics/plot)**(**[**head**](http://www.rdocumentation.org/packages/utils/topics/head)**(cars, input$n)) )**

**}**

**)**

[**runApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/runApp.html)**(app)**

**}**

Run Shiny Application

# Run Shiny Application

[runApp](https://shiny.rstudio.com/reference/shiny/1.3.2/runApp.html)(appDir = [getwd](http://www.rdocumentation.org/packages/base/topics/getwd)(), port = [getOption](http://www.rdocumentation.org/packages/base/topics/options)("shiny.port"),

launch.browser = [getOption](http://www.rdocumentation.org/packages/base/topics/options)("shiny.launch.browser", [interactive](http://www.rdocumentation.org/packages/base/topics/interactive)()),

host = [getOption](http://www.rdocumentation.org/packages/base/topics/options)("shiny.host", "127.0.0.1"), workerId = "",

quiet = FALSE, display.mode = [c](http://www.rdocumentation.org/packages/base/topics/c)("auto", "normal", "showcase"),

test.mode = [getOption](http://www.rdocumentation.org/packages/base/topics/options)("shiny.testmode", FALSE))

### Arguments

|  |  |
| --- | --- |
| **appDir** | The application to run. Should be one of the following:   * A directory containing server.R, plus, either ui.R or a www directory that contains the file index.html. * A directory containing app.R. * An .R file containing a Shiny application, ending with an expression that produces a Shiny app object. * A list with ui and server components. |
| **port** | The TCP port that the application should listen on. If the port is not specified, and the shiny.port option is set (with options(shiny.port = XX)), then that port will be used. Otherwise, use a random port. |
| **launch.browser** | If true, the system's default web browser will be launched automatically after the app is started. Defaults to true in interactive sessions only. This value of this parameter can also be a function to call with the application's URL. |
| **host** | The IPv4 address that the application should listen on. Defaults to the shiny.host option, if set, or "127.0.0.1" if not. See Details. |
| **workerId** | Can generally be ignored. Exists to help some editions of Shiny Server Pro route requests to the correct process. |
| **quiet** | Should Shiny status messages be shown? Defaults to FALSE. |
| **display.mode** | The mode in which to display the application. If set to the value "showcase", shows application code and metadata from a DESCRIPTION file in the application directory alongside the application. If set to "normal", displays the application normally. Defaults to "auto", which displays the application in the mode given in its DESCRIPTION file, if any. |
| **test.mode** | Should the application be launched in test mode? This is only used for recording or running automated tests. Defaults to the shiny.testmode option, or FALSE if the option is not set. |

### Description

Runs a Shiny application. This function normally does not return; interrupt R to stop the application (usually by pressing Ctrl+C or Esc).

### Details

The host parameter was introduced in Shiny 0.9.0. Its default value of "127.0.0.1" means that, contrary to previous versions of Shiny, only the current machine can access locally hosted Shiny apps. To allow other clients to connect, use the value "0.0.0.0" instead (which was the value that was hard-coded into Shiny in 0.8.0 and earlier).

### Examples

**## Not run: ------------------------------------**

**# # Start app in the current working directory**

**# runApp()**

**#**

**# # Start app in a subdirectory called myapp**

**# runApp("myapp")**

**## ---------------------------------------------**

**## Only run this example in interactive R sessions**

**if (**[**interactive**](http://www.rdocumentation.org/packages/base/topics/interactive)**()) {**

[**options**](http://www.rdocumentation.org/packages/base/topics/options)**(device.ask.default = FALSE)**

**# Apps can be run without a server.r and ui.r file**

[**runApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/runApp.html)**(**[**list**](http://www.rdocumentation.org/packages/base/topics/list)**(**

**ui =** [**bootstrapPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/bootstrapPage.html)**(**

[**numericInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/numericInput.html)**('n', 'Number of obs', 100),**

[**plotOutput**](https://shiny.rstudio.com/reference/shiny/1.3.2/plotOutput.html)**('plot')**

**),**

**server = function(input, output) {**

**output$plot <-** [**renderPlot**](https://shiny.rstudio.com/reference/shiny/1.3.2/renderPlot.html)**({** [**hist**](http://www.rdocumentation.org/packages/graphics/topics/hist)**(**[**runif**](http://www.rdocumentation.org/packages/stats/topics/Uniform)**(input$n)) })**

**}**

**))**

**# Running a Shiny app object**

**app <-** [**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(**

**ui =** [**bootstrapPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/bootstrapPage.html)**(**

[**numericInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/numericInput.html)**('n', 'Number of obs', 100),**

[**plotOutput**](https://shiny.rstudio.com/reference/shiny/1.3.2/plotOutput.html)**('plot')**

**),**

**server = function(input, output) {**

**output$plot <-** [**renderPlot**](https://shiny.rstudio.com/reference/shiny/1.3.2/renderPlot.html)**({** [**hist**](http://www.rdocumentation.org/packages/graphics/topics/hist)**(**[**runif**](http://www.rdocumentation.org/packages/stats/topics/Uniform)**(input$n)) })**

**}**

**)**

[**runApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/runApp.html)**(app)**

**}**

Enable Bookmarking in Shiny App:

Enable bookmarking for a Shiny application

[enableBookmarking](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)(store = [c](http://www.rdocumentation.org/packages/base/topics/c)("url", "server", "disable"))

Arguments

|  |  |
| --- | --- |
| **store** | Either "url", which encodes all of the relevant values in a URL, "server", which saves to disk on the server, or "disable", which disables any previously-enabled bookmarking. |

Description

There are two types of bookmarking: saving an application's state to disk on the server, and encoding the application's state in a URL. For state that has been saved to disk, the state can be restored with the corresponding state ID. For URL-encoded state, the state of the application is encoded in the URL, and no server-side storage is needed.

URL-encoded bookmarking is appropriate for applications where there not many input values that need to be recorded. Some browsers have a length limit for URLs of about 2000 characters, and if there are many inputs, the length of the URL can exceed that limit.

Saved-on-server bookmarking is appropriate when there are many inputs, or when the bookmarked state requires storing files.

Details

For restoring state to work properly, the UI must be a function that takes one argument, request. In most Shiny applications, the UI is not a function; it might have the form fluidPage(....). Converting it to a function is as simple as wrapping it in a function, as in function(request) { fluidPage(....) }.

By default, all input values will be bookmarked, except for the values of passwordInputs. fileInputs will be saved if the state is saved on a server, but not if the state is encoded in a URL.

When bookmarking state, arbitrary values can be stored, by passing a function as the onBookmark argument. That function will be passed a ShinySaveState object. The values field of the object is a list which can be manipulated to save extra information. Additionally, if the state is being saved on the server, and the dir field of that object can be used to save extra information to files in that directory.

For saved-to-server state, this is how the state directory is chosen:

* If running in a hosting environment such as Shiny Server or Connect, the hosting environment will choose the directory.
* If running an app in a directory with [runApp](https://shiny.rstudio.com/reference/shiny/1.3.2/runApp.html)(), the saved states will be saved in a subdirectory of the app called shiny\_bookmarks.
* If running a Shiny app object that is generated from code (not run from a directory), the saved states will be saved in a subdirectory of the current working directory called shiny\_bookmarks.

When used with shinyApp(), this function must be called before shinyApp(), or in the shinyApp()'s onStart function. An alternative to calling the enableBookmarking() function is to use the enableBookmarking *argument* for shinyApp(). See examples below.

Examples

**## Only run these examples in interactive R sessions**

**if (**[**interactive**](http://www.rdocumentation.org/packages/base/topics/interactive)**()) {**

**# Basic example with state encoded in URL**

**ui <- function(request) {**

[**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**textInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/textInput.html)**("txt", "Text"),**

[**checkboxInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/checkboxInput.html)**("chk", "Checkbox"),**

[**bookmarkButton**](https://shiny.rstudio.com/reference/shiny/1.3.2/bookmarkButton.html)**()**

**)**

**}**

**server <- function(input, output, session) { }**

[**enableBookmarking**](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)**("url")**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server)**

**# An alternative to calling enableBookmarking(): use shinyApp's**

**# enableBookmarking argument**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server, enableBookmarking = "url")**

**# Same basic example with state saved to disk**

[**enableBookmarking**](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)**("server")**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server)**

**# Save/restore arbitrary values**

**ui <- function(req) {**

[**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**textInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/textInput.html)**("txt", "Text"),**

[**checkboxInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/checkboxInput.html)**("chk", "Checkbox"),**

[**bookmarkButton**](https://shiny.rstudio.com/reference/shiny/1.3.2/bookmarkButton.html)**(),**

[**br**](https://shiny.rstudio.com/reference/shiny/1.3.2/builder.html)**(),**

[**textOutput**](https://shiny.rstudio.com/reference/shiny/1.3.2/textOutput.html)**("lastSaved")**

**)**

**}**

**server <- function(input, output, session) {**

**vals <-** [**reactiveValues**](https://shiny.rstudio.com/reference/shiny/1.3.2/reactiveValues.html)**(savedTime = NULL)**

**output$lastSaved <-** [**renderText**](https://shiny.rstudio.com/reference/shiny/1.3.2/renderText.html)**({**

**if (!**[**is.null**](http://www.rdocumentation.org/packages/base/topics/NULL)**(vals$savedTime))**

[**paste**](http://www.rdocumentation.org/packages/base/topics/paste)**("Last saved at", vals$savedTime)**

**else**

**""**

**})**

[**onBookmark**](https://shiny.rstudio.com/reference/shiny/1.3.2/onBookmark.html)**(function(state) {**

**vals$savedTime <-** [**Sys.time**](http://www.rdocumentation.org/packages/base/topics/Sys.time)**()**

**# state is a mutable reference object, and we can add arbitrary values**

**# to it.**

**state$values$time <- vals$savedTime**

**})**

[**onRestore**](https://shiny.rstudio.com/reference/shiny/1.3.2/onBookmark.html)**(function(state) {**

**vals$savedTime <- state$values$time**

**})**

**}**

[**enableBookmarking**](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)**(store = "url")**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server)**

**# Usable with dynamic UI (set the slider, then change the text input,**

**# click the bookmark button)**

**ui <- function(request) {**

[**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**sliderInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/sliderInput.html)**("slider", "Slider", 1, 100, 50),**

[**uiOutput**](https://shiny.rstudio.com/reference/shiny/1.3.2/htmlOutput.html)**("ui"),**

[**bookmarkButton**](https://shiny.rstudio.com/reference/shiny/1.3.2/bookmarkButton.html)**()**

**)**

**}**

**server <- function(input, output, session) {**

**output$ui <-** [**renderUI**](https://shiny.rstudio.com/reference/shiny/1.3.2/renderUI.html)**({**

[**textInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/textInput.html)**("txt", "Text", input$slider)**

**})**

**}**

[**enableBookmarking**](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)**("url")**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server)**

**# Exclude specific inputs (The only input that will be saved in this**

**# example is chk)**

**ui <- function(request) {**

[**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**passwordInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/passwordInput.html)**("pw", "Password"), # Passwords are never saved**

[**sliderInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/sliderInput.html)**("slider", "Slider", 1, 100, 50), # Manually excluded below**

[**checkboxInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/checkboxInput.html)**("chk", "Checkbox"),**

[**bookmarkButton**](https://shiny.rstudio.com/reference/shiny/1.3.2/bookmarkButton.html)**()**

**)**

**}**

**server <- function(input, output, session) {**

[**setBookmarkExclude**](https://shiny.rstudio.com/reference/shiny/1.3.2/setBookmarkExclude.html)**("slider")**

**}**

[**enableBookmarking**](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)**("url")**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server)**

**# Update the browser's location bar every time an input changes. This should**

**# not be used with enableBookmarking("server"), because that would create a**

**# new saved state on disk every time the user changes an input.**

**ui <- function(req) {**

[**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**textInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/textInput.html)**("txt", "Text"),**

[**checkboxInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/checkboxInput.html)**("chk", "Checkbox")**

**)**

**}**

**server <- function(input, output, session) {**

[**observe**](https://shiny.rstudio.com/reference/shiny/1.3.2/observe.html)**({**

**# Trigger this observer every time an input changes**

[**reactiveValuesToList**](https://shiny.rstudio.com/reference/shiny/1.3.2/reactiveValuesToList.html)**(input)**

**session$doBookmark()**

**})**

[**onBookmarked**](https://shiny.rstudio.com/reference/shiny/1.3.2/onBookmark.html)**(function(url) {**

[**updateQueryString**](https://shiny.rstudio.com/reference/shiny/1.3.2/updateQueryString.html)**(url)**

**})**

**}**

[**enableBookmarking**](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)**("url")**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server)**

**# Save/restore uploaded files**

**ui <- function(request) {**

[**fluidPage**](https://shiny.rstudio.com/reference/shiny/1.3.2/fluidPage.html)**(**

[**sidebarLayout**](https://shiny.rstudio.com/reference/shiny/1.3.2/sidebarLayout.html)**(**

[**sidebarPanel**](https://shiny.rstudio.com/reference/shiny/1.3.2/sidebarPanel.html)**(**

[**fileInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/fileInput.html)**("file1", "Choose CSV File", multiple = TRUE,**

**accept =** [**c**](http://www.rdocumentation.org/packages/base/topics/c)**(**

**"text/csv",**

**"text/comma-separated-values,text/plain",**

**".csv"**

**)**

**),**

**tags$**[**hr**](https://shiny.rstudio.com/reference/shiny/1.3.2/builder.html)**(),**

[**checkboxInput**](https://shiny.rstudio.com/reference/shiny/1.3.2/checkboxInput.html)**("header", "Header", TRUE),**

[**bookmarkButton**](https://shiny.rstudio.com/reference/shiny/1.3.2/bookmarkButton.html)**()**

**),**

[**mainPanel**](https://shiny.rstudio.com/reference/shiny/1.3.2/mainPanel.html)**(**

[**tableOutput**](https://shiny.rstudio.com/reference/shiny/1.3.2/tableOutput.html)**("contents")**

**)**

**)**

**)**

**}**

**server <- function(input, output) {**

**output$contents <-** [**renderTable**](https://shiny.rstudio.com/reference/shiny/1.3.2/renderTable.html)**({**

**inFile <- input$file1**

**if (**[**is.null**](http://www.rdocumentation.org/packages/base/topics/NULL)**(inFile))**

[**return**](http://www.rdocumentation.org/packages/base/topics/function)**(NULL)**

**if (**[**nrow**](http://www.rdocumentation.org/packages/base/topics/nrow)**(inFile) == 1) {**

[**read.csv**](http://www.rdocumentation.org/packages/utils/topics/read.table)**(inFile$datapath, header = input$header)**

**} else {**

[**data.frame**](http://www.rdocumentation.org/packages/base/topics/data.frame)**(x = "multiple files")**

**}**

**})**

**}**

[**enableBookmarking**](https://shiny.rstudio.com/reference/shiny/1.3.2/enableBookmarking.html)**("server")**

[**shinyApp**](https://shiny.rstudio.com/reference/shiny/1.3.2/shinyApp.html)**(ui, server)**

**}**

So according to the documentation we should rarely call shinyApp() directly, and use only runApp() instead. But using runApp() is impossible on RStudio platforms, as they print an error that looks like this:

Loading aaaa

Error in shiny::runApp(system.file("app", package = "aaaa")) :

Can't call `runApp()` from within `runApp()`. If your application code contains `runApp()`, please remove it.

Calls: runApp ... eval -> eval -> ..stacktraceon.. -> run\_app ->

**Side note 2**

I’ve tried to make this benchmark as reproducible as possible, so **feel free to run it and see if you get the same results** 

The package is named “aaaa” (so it won’t conflict with any other package (I hope), and can be found [here](https://github.com/ColinFay/golem4bench). It contains the golem skeleton with the functions listed below.

**Looking for the best implementation**

Anyway, let’s try to find the best implementation. The idea is that our implementation of run\_app() should:

* **work on the maximum number of services** (Locally + Docker + RStudio products), and this with minimal tweaking (one implementation to rule them all would be best).
* **Be able to read options from the global environment**, so that for example we can use the global golem.app.prod variable from inside the server and UI, or global options / env var defined in the service.
* Be able to read **options from the local function environment**, so we can pass arguments to the run\_app() function.

**Benchmark conditions**

* Previous version of {golem} (0.0.1.600)

**Content of the app\_ui function:**

app\_ui <- function() {

tagList(

fluidPage(

h1("aaaa"),

h3( "global options:" ),

verbatimTextOutput("global"),

h3( "function options:" ),

verbatimTextOutput("shinycall")

)

)

}

**Content of the app\_server function:**

app\_server <- function(input, output,session) {

output$global <- renderPrint({

# Global options

getOption('golem.pkg.name')

})

output$shinycall <- renderPrint({

# Local options

getOption('shinycall')

})

}

**Various run\_app implementations**

We’ll use this function to benchmark the three functions (shinyApp(), runApp(), and shinyAppDir().

run\_app <- function(

with = c("shinyApp", "runApp", "shinyAppDir")

) {

with <- match.arg(with)

# Here, we set local options so we can pass

# arguments to the run\_app() function

options("shinycall" = with)

on.exit(

options("shinycall" = NULL)

)

if (with == "shinyApp"){

shiny::shinyApp(ui = app\_ui(), server = app\_server)

} else if (with == "runApp") {

shiny::runApp(system.file("app", package = "aaaa"))

} else {

shiny::shinyAppDir(system.file("app", package = "aaaa"))

}

}

**In our results, runApp is:**

run\_app( "runApp" )

**shinyApp is:**

run\_app( "shinyApp" )

**shinyAppDir is**

run\_app( "shinyAppDir" )

**Launch contexts**

**Local launch**

You can run this in your console, here in RStudio.

# Set options here

options( "golem.pkg.name" = "aaa")

# Detach all loaded packages and clean your environment

golem::detach\_all\_attached()

# rm(list=ls(all.names = TRUE))

# Document and reload your package

golem::document\_and\_reload()

# Run the application

aaaa::run\_app(with = "runApp")

**The Dockerfile for local test is**

FROM rocker/tidyverse:3.6.0

RUN R -e 'install.packages("remotes")'

RUN R -e 'remotes::install\_cran("shiny")'

COPY aaaa\_\*.tar.gz /app.tar.gz

RUN R -e 'remotes::install\_local("/app.tar.gz")'

EXPOSE 80

CMD R -e "options('shiny.port'=1234,shiny.host='0.0.0.0', 'golem.pkg.name' = 'aaa');aaaa::run\_app( 'runApp' )" # also with shinyApp & shinyAppDir

You can find this Dockerfile in the inst/dockerfilelocal folder of the golem4bench repo. Before launching it, **you have to document, and build your package** with devtools::build(path = "inst/dockerfilelocal/"),

The full thing can be launched with:

R -e "devtools::build(path = 'inst/dockerfilelocal/')" \

&& cd inst/dockerfilelocal/ \

&& docker build -t aaa . \

&& docker run --name aaaa -p 1234:1234 -d aaa \

&& sleep 2 \

&& open http://0.0.0.0:1234

Then, stay in the folder, change the "runApp" arg in the Dockerfile to "shinyApp", **rebuild and relaunch** from the docker build line. Then again with "shinyAppDir". Of course, don’t forget to docker kill aaa && docker rm aaa between each iteration.

**RStudio products 1/2: the app.R script**

pkgload::load\_all()

options( "golem.pkg.name" = "aaa" )

run\_app( "runApp" ) # also with shinyApp & shinyAppDir

Each three versions (i.e the three versions of run\_app()) of this file will be deployed to:

* **local Shiny Server** (copied inside the Docker)
* **ThinkR internal RStudio Connect** (sent with rsconnect::deployApp())
* **ThinkR’s shinyapps.io account** (sent with rsconnect::deployApp())

**This file should be put at the root of your package.**

**RStudio products 2/2: Setting a Shiny server for testing**

This Dockerfile can be found in the inst/dockerfileshinyserver folder of the package.

FROM rocker/shiny:3.6.0

RUN R -e 'install.packages("remotes")'

RUN R -e 'remotes::install\_cran("shiny")'

RUN apt-get update && apt-get install libssl-dev libxml2-dev -y

RUN R -e 'remotes::install\_cran("attachment")'

COPY . /srv/shiny-server/aaaa

RUN cd /srv/shiny-server/aaaa && R -e "attachment::install\_from\_description()"

From the root of the package:

mv inst/dockerfileshinyserver/Dockerfile Dockerfile \

&& docker build -t plop . \

&& docker run --name plop -p 3838:3838 -d plop \

&& sleep 2 \

&& open http://0.0.0.0:3838/aaaa

Then, change for "shinyApp" and "shinyAppDir" in the app.R file, then rerun the docker build. Don’t forget to kill & rm files between each iteration, and to mv back the Dockerfile where it belongs.

**Results**

Global options are the one defined outside of run\_app(), local options are the one defined inside the run\_app().

: the app launches   
: the app doesn’t launch  
: the global options are read   
: the global options are not read   
: the function options are read  
: the function options are not read

| **Where** | **runApp** | **shinyApp** | **shinyAppDir** |
| --- | --- | --- | --- |
| Locally | 🚀📗📒 | 🚀📗⛔ | 🚀📗⛔ |
| Docker | 🚀📗📒 | 🚀📗⛔ | 🚀📗⛔ |
| Connect | 💥❌⛔ | 🚀📗⛔ | 🚀📗⛔ |
| shinyApps.io | 💥❌⛔ | 🚀📗⛔ | 🚀📗⛔ |
| ShinyServer | 💥❌⛔ | 🚀📗⛔ | 🚀📗⛔ |

So to sum up :

* Docker containers don’t get local options from the functions unless called with runApp(). Which you can verify with running in any terminal: R -e "options('shiny.port'=1234,shiny.host='0.0.0.0', 'golem.pkg.name' = 'aaa');aaaa::run\_app( 'runApp' )"
* runApp() fails on RStudio Product.
* RStudio products don’t get local options with any solution (we’ll see in a minute that’s because we can’t use runApp().

**Where do we go from there?**

So, why this different behaviours? Actually, it’s because of what shinyApp() and shinyAppDir() return, compared to runApp(). If we look at the source code of shinyApp(), the last line of code looks like this:

> shiny::shinyApp

function (ui = NULL, server = NULL, onStart = NULL, options = list(),

uiPattern = "/", enableBookmarking = NULL)

{

[...]

structure(list(httpHandler = httpHandler, serverFuncSource = serverFuncSource,

onStart = onStart, options = options, appOptions = appOptions),

class = "shiny.appobj")

}

We can see that the last thing returned by the function is a structure of class shiny.appobj, whereas the runApp() returns a running process. So the “launch” of the app with shinyApp() is not the same the the one from runApp()— the first returns an object, while the second returns a process. So the launch of the app, with shinyApp(), is actually done with print.shiny.appobj. Which is why if you do a <- shinyAppDir(appDir = "inst/app/"), you’ll not get the app running until you try to print a. Which also explains why the local options (defined inside the function) are not read: with shinyApp(), the function does return an object, so the function has ended, and the options defined there are not accessible anymore.

Why is it a good news? Let’s have a look at shiny:::print.shiny.appobj:

> shiny:::print.shiny.appobj

function (x, ...)

{

opts <- x$options %OR% list()

opts <- opts[names(opts) %in% c("port", "launch.browser",

"host", "quiet", "display.mode", "test.mode")]

args <- c(list(quote(x)), opts)

do.call("runApp", args)

}

So here the cool thing is that we can hack the x passed to the print method to add golem.options inside it, i.e. in the appOptions of the app object. Hence:

with\_golem\_options <- function(app, golem\_opts){

app$appOptions$golem\_options <- golem\_opts

app

}

and

run\_app <- function(...) {

with\_golem\_options(

app = shinyApp(ui = app\_ui(), server = app\_server),

golem\_opts = list(...)

)

}

And with a full app:

library(shiny)

options("golem.app.name" = "aaa")

get\_golem\_options <- function(which = NULL){

if (is.null(which)){

getShinyOption("golem\_options")

} else {

getShinyOption("golem\_options")[[which]]

}

}

with\_golem\_options <- function(app, golem\_opts){

app$appOptions$golem\_options <- golem\_opts

app

}

app\_ui <- function() {

tagList(

fluidPage(

verbatimTextOutput("all"),

verbatimTextOutput("opt"),

verbatimTextOutput("glob")

)

)

}

app\_server <- function(input, output,session) {

output$all <- renderPrint({ get\_golem\_options() })

output$opt <- renderPrint({ get\_golem\_options("a") })

output$glob <- renderPrint({ getOption("golem.app.name") })

}

run\_app <- function(...) {

with\_golem\_options(

app = shinyApp(ui = app\_ui(), server = app\_server),

golem\_opts = list(...)

)

}

run\_app(a = "pouet", b = "bing")

And the good news is… it works everywhere 

So thanks to this little hack, you are now able to use the run\_app() function from {golem} everywhere. And now, when you build your package, you can use arguments with the run\_app() function, and use them with get\_golem\_options(). Global options are, as usual, available with getOptions().

The other little cool hack? When you generate an app.R file for RStudio products with golem::add\_rstudioconnect\_file, golem::add\_shinyappsio\_file, or golem::add\_shinyserver\_file, we left a small ShinyApp in a comment, so you can use the nice blue button to deploy in just one click (yes, apparently it seems that every time RStudio sees a ShinyApp in the text, it shows this little button )