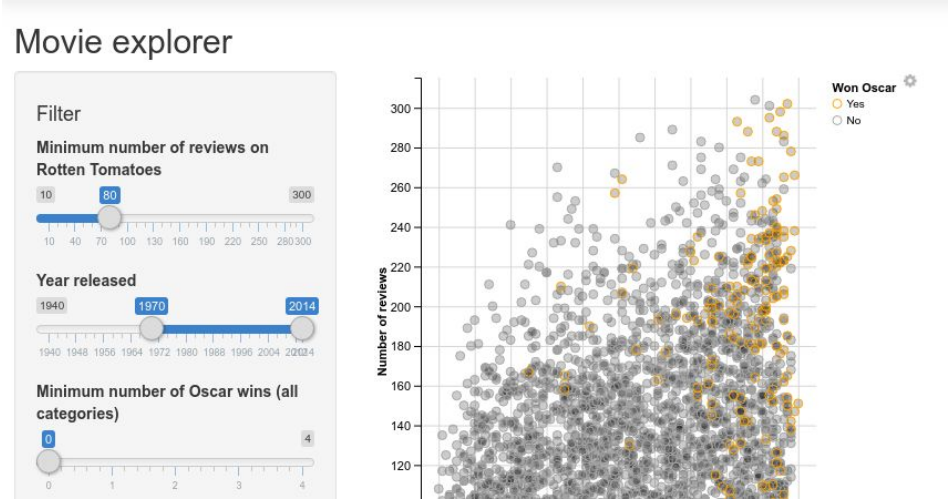
Introduction to R Shiny

R Shiny

R Shiny is a packages that facilitates the creation of interactive web apps or dashboards using R.

Only requires R, although some knowledge of HTML can be useful at times.

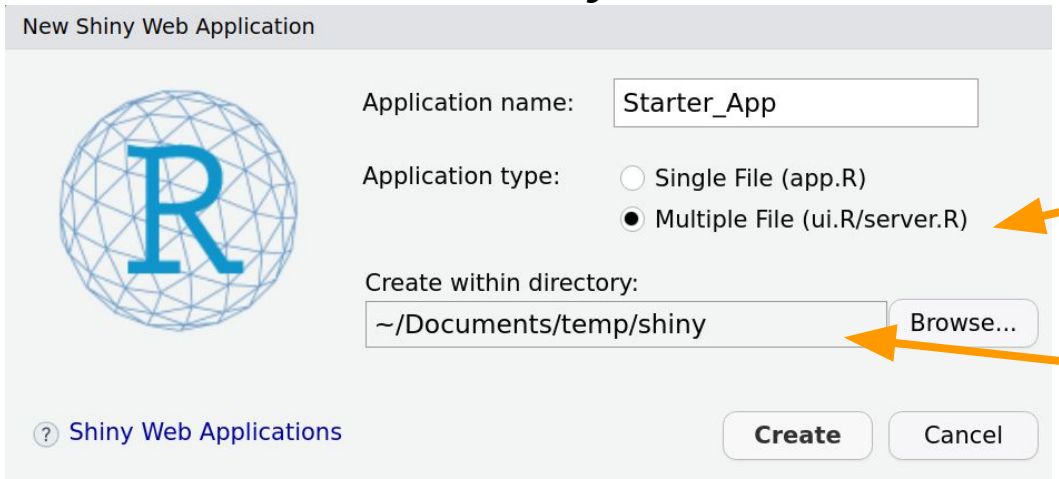
Check out the R Shiny gallery at https://shiny.posit.co/r/gallery/



R Shiny

To create a Shiny app from RStudio, choose File > New File > Shiny Web App

Browse to or create a new directory for your app. After creating it, you probably want to change your working directory to the location of your app and make a data folder in that directory.



Choose Multiple File for the Application type.

This will create a new directory for the app at the location you select.

Components of a Shiny App

Every Shiny App consists of two primary components:

- **ui: “**Front-End”
  - Controls layout and appearance
  - What the user sees

- **server:** “Back-End”
  - Contains instructions for building your app
  - What gets updated/run as the user interacts with the interface

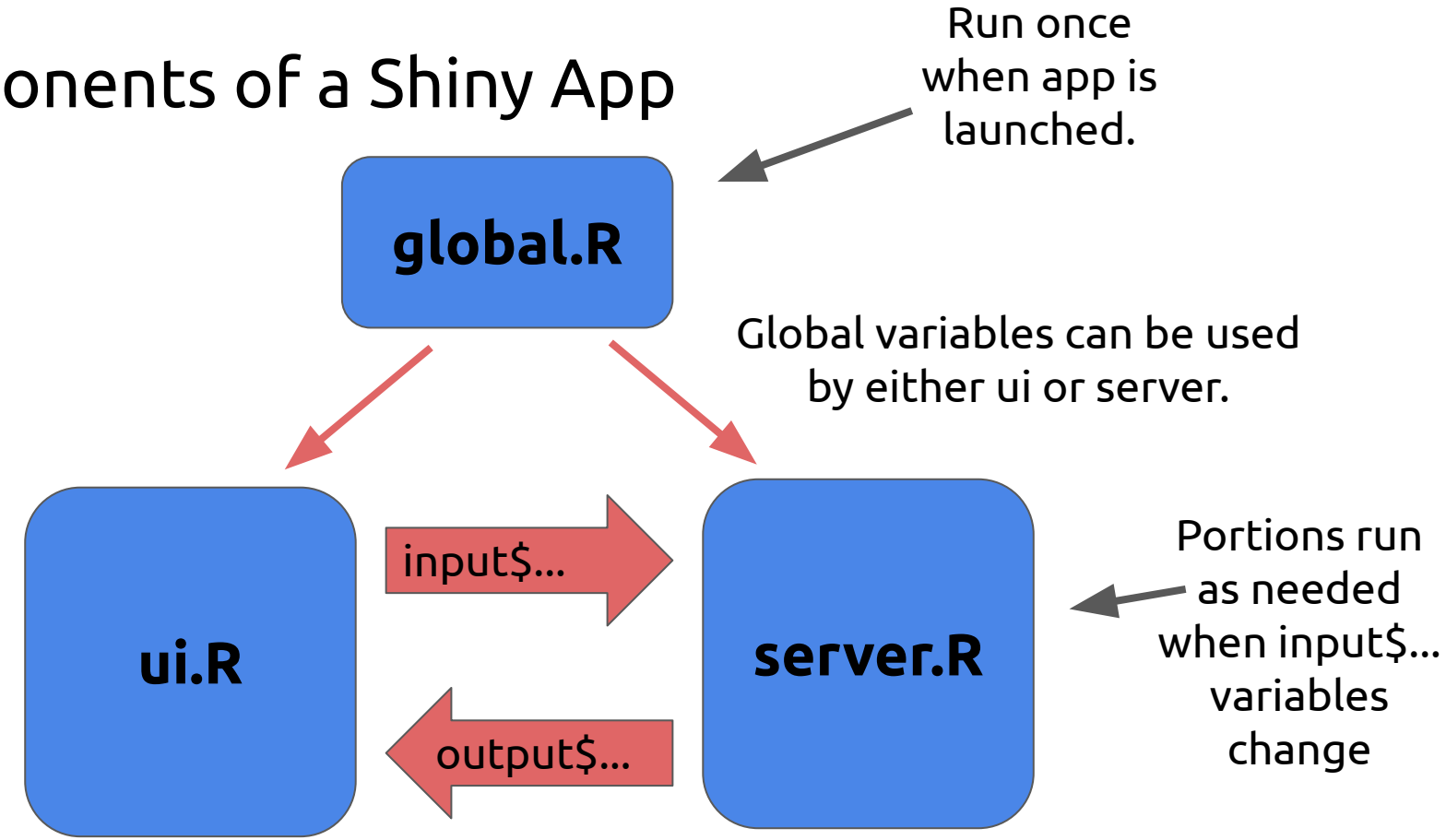
Components of a Shiny App

It is often useful to include a third component:

global: Objects visible to both server and ui
Useful for loading in datasets and libraries

The code in global.R is run once before the app starts.

Components of a Shiny App



ui - Widgets

Widgets can be used to allow the user to interact with your app.

All widgets require a **name** (used to access its value) and a **label** (which the user sees).

```
sliderInput("bins","Number of bins:", min = 1, max = 50, value = 30)
```

In the **server** portion of our app, we will be able to access the value of this widget by using *input$bins*.

ui - Widgets

There are a variety of types of widgets:

- *actionButton*
- *selectInput*
- *sliderInput*
- *textInput*

See the Shiny Widgets Gallery for more information:
https://shiny.posit.co/r/gallery/widgets/widget-gallery/

server - Reactive Elements

Reactive elements automatically respond when the user modifies a widget value.

Can be used to update a plot, filter a dataframe, call a function, or other tasks.

Shiny can determine what the output from a reactive expression depends on and only rerun it if the input changes.

If you modularize your code into a series of reactive function calls, you can make your app more efficient.

See https://shiny.posit.co/r/getstarted/shiny-basics/lesson6/

server - Reactive Elements

Two steps to add a reactive element:

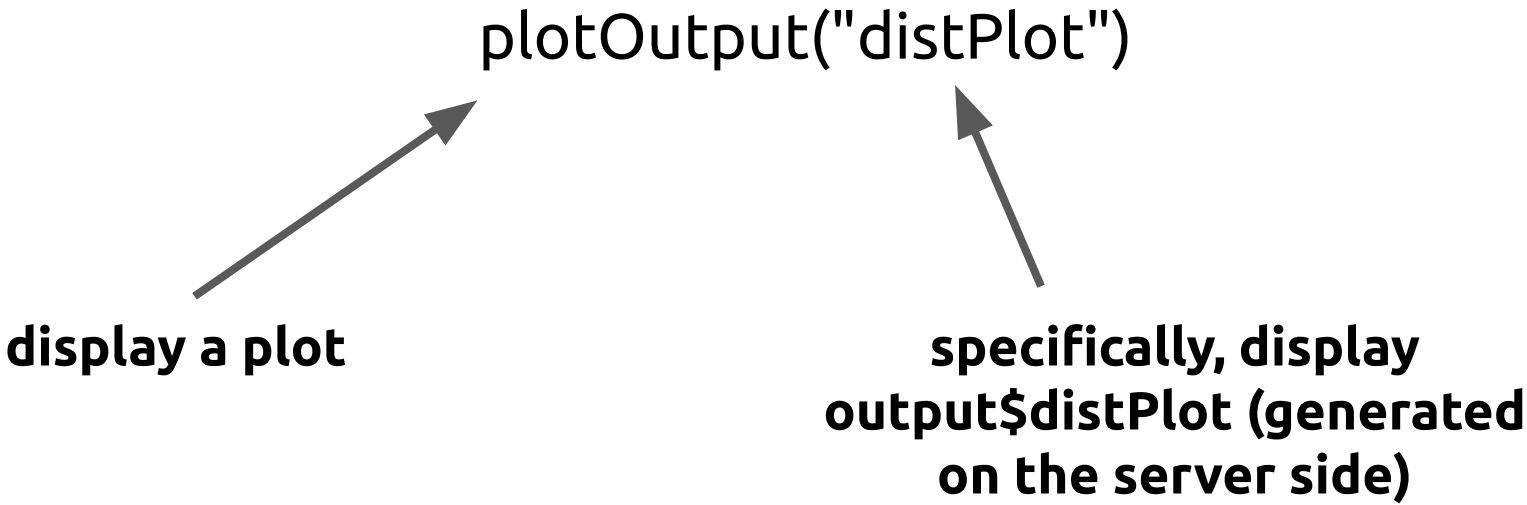
1. Add an R object to your user interface
2. Tell Shiny how to build the object in the server function. The object will be reactive if the code that builds it uses a widget value.

ui - Reactive Elements

There are a number of different types of objects you can add to your interface using *Output.
These functions are called in the **ui** side.

Output function	Creates
`dataTableOutput`	DataTable
`htmlOutput`	raw HTML
`imageOutput`	image
`plotOutput`	plot
`tableOutput`	table
`textOutput`	text
`uiOutput`	raw HTML
`verbatimTextOutput`	text

ui - Reactive Elements



server - Reactive Elements

For every output function you call, there should be a corresponding **render*** call on the **server** side:

render function	creates
`renderDataTable`	DataTable
`renderImage`	images (saved as a link to a source file)
`renderPlot`	plots
`renderPrint`	any printed output
`renderTable`	data frame, matrix, other table like structures
`renderText`	character strings
`renderUI`	a Shiny tag object or HTML

server - Reactive Elements

how to reference this plot in ui

This must contain an expression that generates a plot, contained inside the braces {}

```
output$distPlot <- renderPlot({
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
```

refers to the value from the sliderInput

ui - Reactive Elements

The ({ … }) looks funny, but just remember that the portion inside the {...} is R code. You can also add additional arguments inside the (), such as height and width.

```
output$distPlot <- renderPlot({
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
    })
```

Resources for Learning Shiny

RStudio's official tutorial:
https://shiny.posit.co/r/getstarted/shiny-basics/lesson1/

DataCamp Courses:
https://www.datacamp.com/courses/building-dashboards-with-shinydashboard
https://www.datacamp.com/courses/building-web-applications-in-r-with-shiny-case-studies

The Gallery (A lot of apps will let you see the code):
https://shiny.posit.co/r/gallery/

Additional Shiny Resources

Mastering Shiny (Free eBook): https://mastering-shiny.org/index.html

R Shiny Cheatsheet: https://shiny.posit.co/r/articles/start/cheatsheet/

Curated List of Shiny Extensions and Resources:

https://github.com/nanxstats/awesome-shiny-extensions

https://github.com/grabear/awesome-rshiny

Resources for Learning Shiny

Shiny comes with some built-in examples as well:

```
runExample("01_hello")      # a histogram
runExample("02_text")       # tables and data frames
runExample("03_reactivity") # a reactive expression
runExample("04_mpg")        # global variables
runExample("05_sliders")    # slider bars
runExample("06_tabsets")    # tabbed panels
runExample("07_widgets")    # help text and submit buttons
runExample("08_html")       # Shiny app built from HTML
runExample("09_upload")     # file upload wizard
runExample("10_download")   # file download wizard
runExample("11_timer")      # an automated timer
```

ui

Every Shiny app needs a layout - see https://shiny.posit.co/r/articles/build/layout-guide/

Examples:

- *fluidPage* (The one that is used when creating a new app in RStudio)
  - Can add other elements such as a *titlePanel*, *sidebarLayout*, *fluidRow*, or *column*
  - Use a *tabsetPanel* for multiple tabs
- *navbarPage* for multiple tabs
- *shinydashboard*

ui Organization

There are a number of elements that can be used to keep the contents of your ui organized.

fluidRow() - Elements within a fluidRow will appear on the same line

column() - Determines how much horizontal space an element will take up (between 1 and 12)

box() - A basic container to hold content. Width between 1 and 12, with respect to its container. (Part of the *shinydashboard* library)

ui Organization

Recommendation: Sketch out the layout that you want for your app. Then figure out how to put it together with fluidRows, columns, and boxes.

Make liberal use of Reindent Lines (Ctrl + I) to make sure that you have the right number of parentheses.

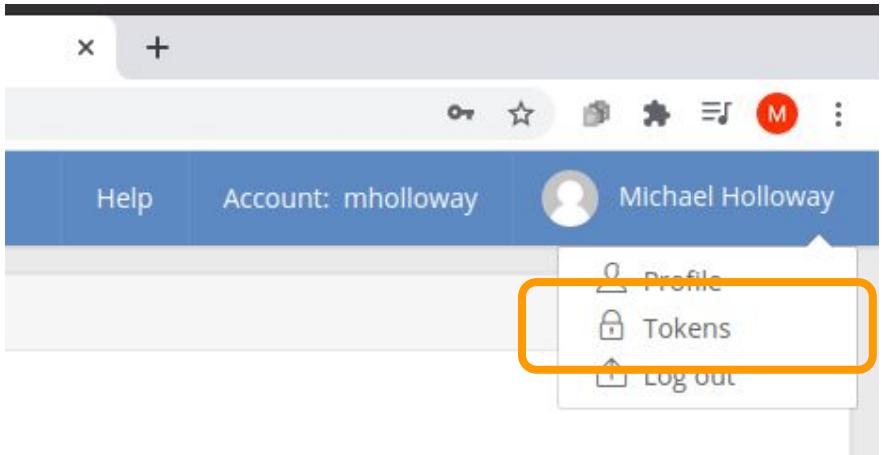
Deploying Your Shiny App

- You will need to create a free account with shinyapps.io

- Also, you will need to install the *rsconnect* package.

Deploying Your Shiny App

Log in to your shinyapps.io account and click on “Tokens” in the dropdown in the upper right.



Deploying Your Shiny App

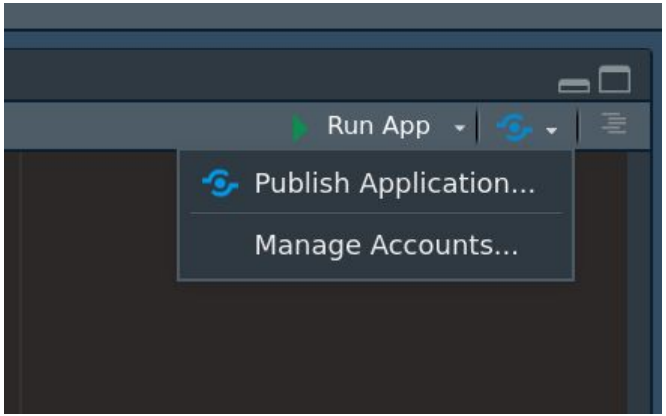
Then click “Show” followed by “Copy to clipboard”.

This will give you a command starting with `rsconnect::setAccountInfo(...`

Copy this command over to RStudio and run it.

Deploying Your Shiny App

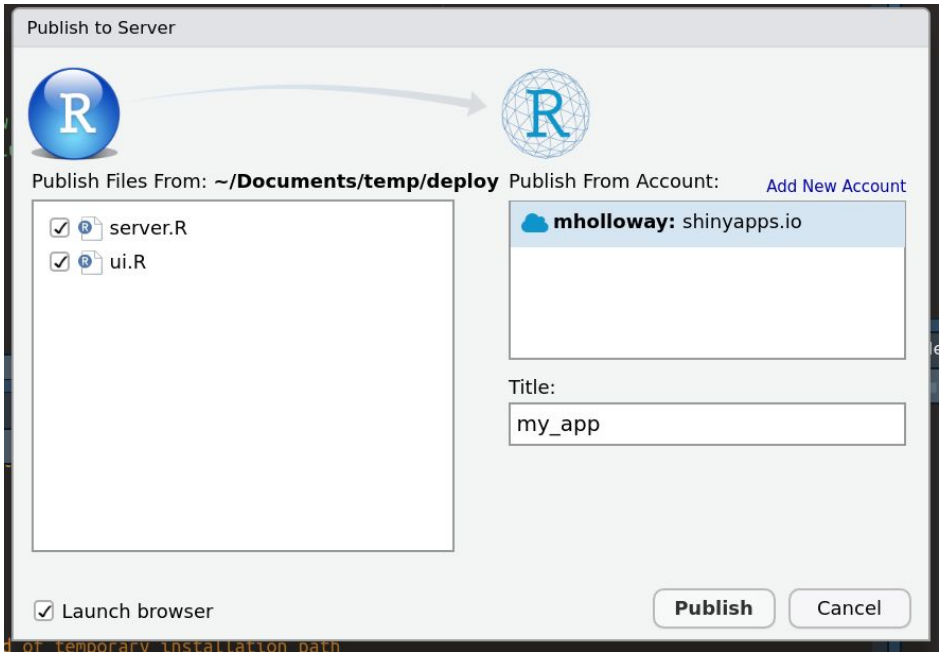
Now that *rsconnect* is configured, you can publish your app. Open one of the .R files for you app. In the upper right corner, there is a “Publish Application...” button.



Deploying Your Shiny App

In the dialog box that pops up, make sure that all necessary files are selected (including global.R and any datafiles needed).

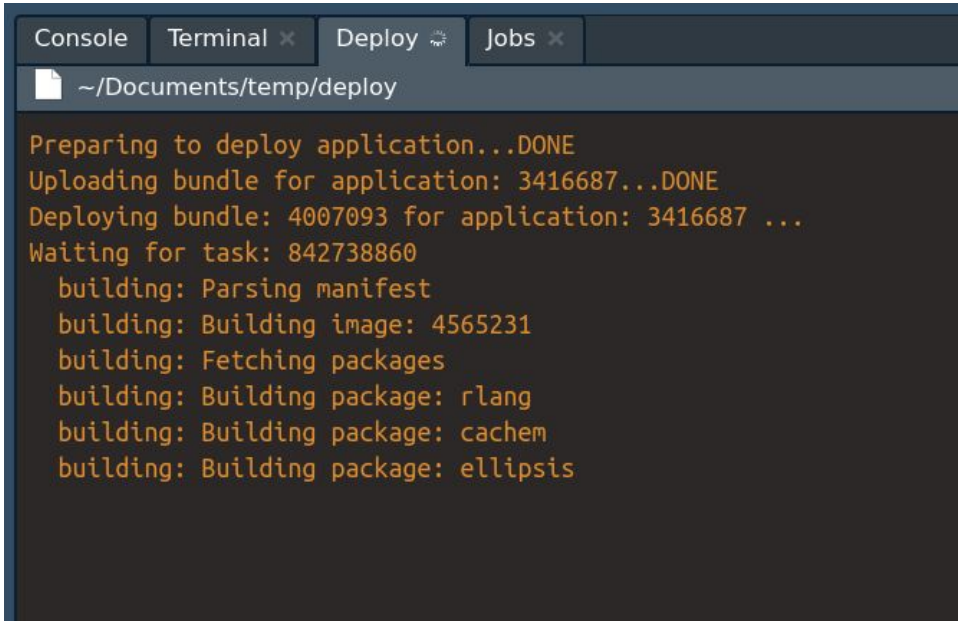
Also, make sure that you are happy with the title.



Deploying Your Shiny App

You'll have to wait a few minutes or so as it builds all the needed packages.

There is a chance this will fail at this step if you try and deploy a large data file.



Deploying Your Shiny App

Once deployed, you can check the status of your apps by logging into your shinyapps.io account.

Here, you can also check the logs if something is not functioning as expected. A common source of problems is if you app uses too much memory and need to be refactored to be more efficient.

It is possible to upgrade your shinyapps.io plan to use more hardware or host your application elsewhere.