[R]eady for production: shiny in production

Martin Schneider

eoda
daten ▸ wissen ▸ nutzen

shiny is awesome, aber …

…Data Science vs. Software Engineering

Data Science

- Agilität
- Wunsch nach Bleeding Edge Technologie
- Gestaltungsfreiheit

Data Science

≠

Software Engineering

…plötzlich Produktiv

Upsi, die App geht produktiv!!!

Was bedeutet es eigentlich in Produktion zu sein?

“Software environments that are used and relied on by

real users, with real consequences if things go wrong.” (Joe Cheng – RStudio)

Anforderungen an eine produktive shiny App

Läuft: keine (signifikante) downtime

Sicher: .. vor unauthorizierten Zugriffen

Korrekt: tut was es soll

Performant: möglichst schnell, skalierbar für erwarteten Traffic

Welche Aspekte sind für Apps in Produktion wichtig?

- Usability
- Wartbar
- Tests
- Loadbalancing

Usability

- Wer ist die Zielgruppe der App?
- Was ist den Usern wichtig?
- Was nicht?
- Wie viele Schritte sind nötig um zum Ziel zu gelangen?
- Sind die Feature der App selbst erklärend?
- Kann man nichts (wenig) falsch machen?

Tools

- **RStudio Connect** – On-premises Shiny serving with push-button deployment

https://www.rstudio.com/products/connect/

- **shiny Modules**

https://shiny.rstudio.com/articles/modules.html

- **shinytest** – Automated UI testing for Shiny

https://rstudio.github.io/shinytest/

- **shinyloadtest** – Load testing for Shiny

https://rstudio.github.io/shinyloadtest/

- **profvis** – Profiler for R

https://rstudio.github.io/profvis/

- **Plot caching** – Dramatically speed up repeated plots

http://shiny.rstudio.com/articles/plot-caching.html

- **Async** – Last resort technique for dealing with slow operations

https://rstudio.github.io/promises/

)

Wartbarkeit – Module

```
# Module UI function
csvFileInput <- function(id, label = "CSV file") {
  # Create a namespace function using the provided id
  ns <- NS(id)

  tagList(
    fileInput(ns("file"), label),
    checkboxInput(ns("heading"), "Has heading"),
    selectInput(ns("quote"), "Quote", c(
      "None" = "",
      "Double quote" = "\"",
      "Single quote" = "'"
    ))
  )
}
```

```
library(shiny)

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      csvFileInput("datafile", "User data (.csv format)")
    ),
    mainPanel(
      dataTableOutput("table")
    )
  )
)

server <- function(input, output, session) {
  datafile <- callModule(csvFile, "datafile",
    stringsAsFactors = FALSE)

  output$table <- renderDataTable({
    datafile()
  })
}

shinyApp(ui, server)
```

)

Wartbarkeit - Module

```
# Module server function
csvFile <- function(input, output, session, stringsAsFactors) {
  # The selected file, if any
  userFile <- reactive({
    # If no file is selected, don't do anything
    validate(need(input$file, message = FALSE))
    input$file
  })

  # The user's data, parsed into a data frame
  dataframe <- reactive({
    read.csv(userFile()$datapath,
      header = input$heading,
      quote = input$quote,
      stringsAsFactors = stringsAsFactors)
  })

  # We can run observers in here if we want to
  observe({
    msg <- sprintf("File %s was uploaded", userFile()$name)
    cat(msg, "\n")
  })

  # Return the reactive that yields the data frame
  return(dataframe)
}
```

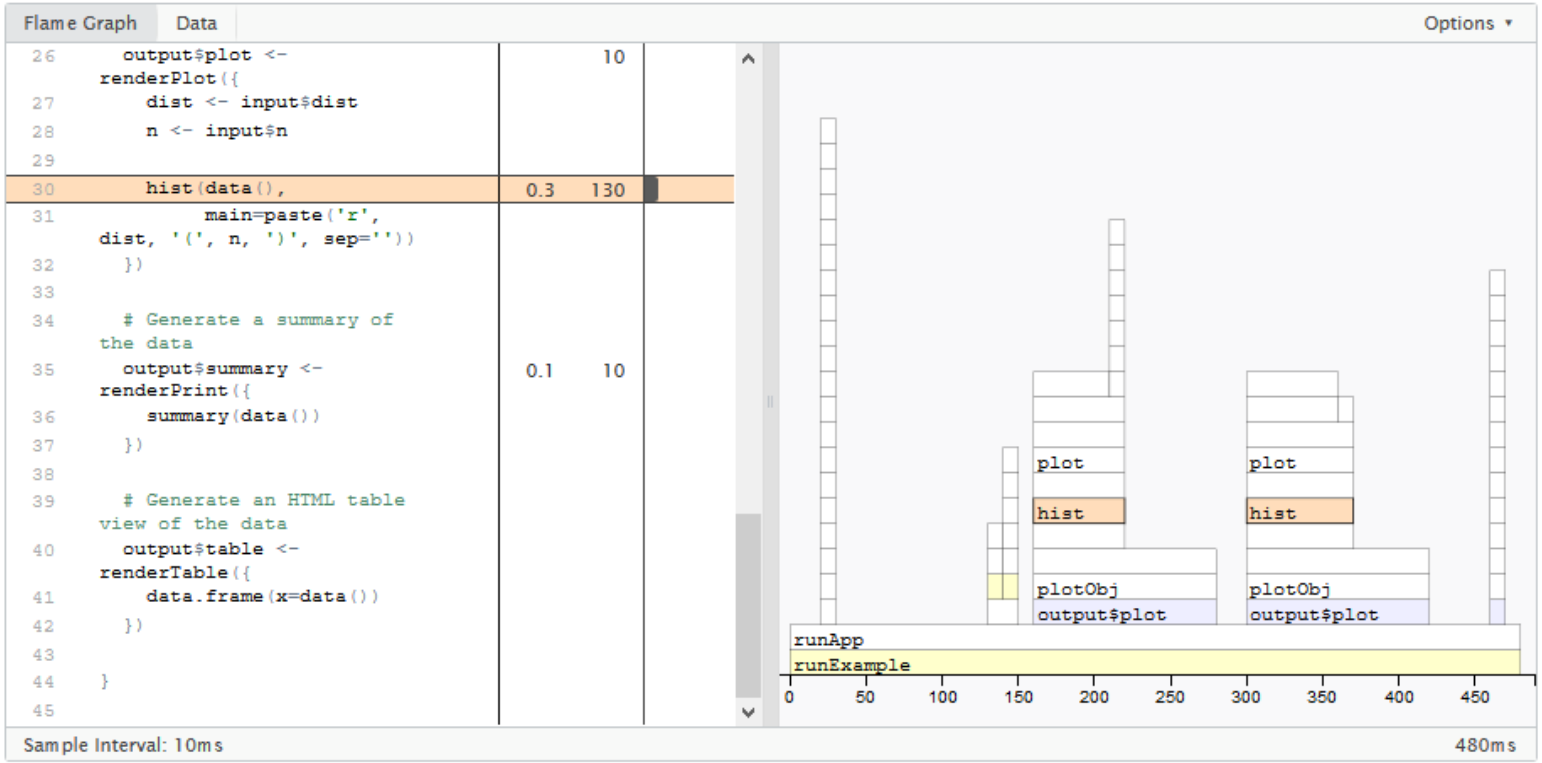
```
library(shiny)

ui <- fluidPage(
  sidebarLayout(
    sidebarPanel(
      csvFileInput("datafile", "User data (.csv format)")
    ),
    mainPanel(
      dataTableOutput("table")
    )
  )
)

server <- function(input, output, session) {
  datafile <- callModule(csvFile, "datafile",
    stringsAsFactors = FALSE)

  output$table <- renderDataTable({
    datafile()
  })
}

shinyApp(ui, server)
```

shinytest

1. Mit shinytest lässt sich eine typische Usersession aufnehmen
2. Während der session wird der Zustand der App als snapshot festgehalten
3. Aufgenommene Sessions lassen sich wiedergeben
4. Während der Wiedergabe wird der aktuellen Zustand mit entsprechenden Snapshots verglichen.

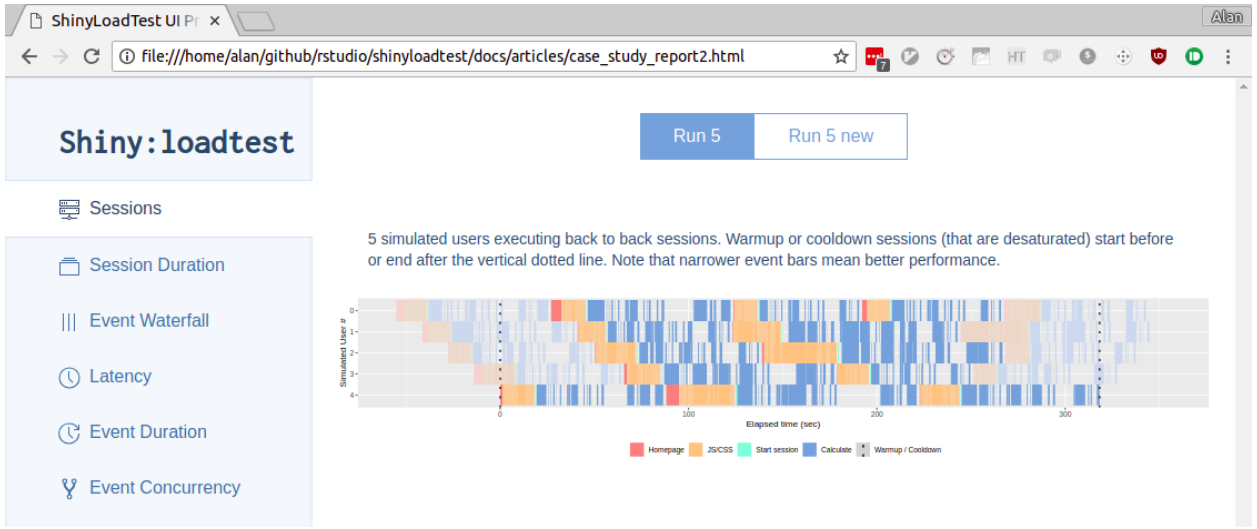
https://rstudio.github.io/shinytest/articles/shinytest.html

profvis



https://rstudio.github.io/profvis/examples.html#example-3---profiling-a-shiny-application

Load Tests



https://rstudio.github.io/shinyloadtest/

Mit der Hilfe von **shinyloadtest** und **shinycannon** kann die App unter **Last** simuliert werden

Workflow

1. App deployen
2. Beispiel Session mit shinyloadtest aufzeichnen
3. Playback unter simulierter Last mit shinycannon
4. Ergebnisse des Lasttests mit shinyloadtest simulieren

Optimierungs Potentiale

1. Wenn möglich Code außerhalb der App laufen lassen -> z.B. ETL Prozesss
2. Code optimieren -> z.B. mit der Hilfe von profvis
3. Caching -> ab shiny 1.2.0
4. Asynchrone Programmierung -> mit promises

ETL mit Rstudio connect

Ein ETL Prozess, lässt sich unter Rstudio connect mit dem markdown scheduling Feature umsetzen

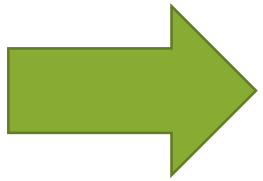
https://docs.rstudio.com/connect/1.6.2/user/r-markdown.html

Plot caching

Ein Bottleneck in shiny Apps sind oft Plots

Wenn absehbar ist, das User die gleichen Plots abrufen, empfiehlt es sich diese zu cachen!

```
output$plot <- renderPlot({
  ggplot(diamonds, aes(
    carat, price, color = !!input$color_by
  )) +
    geom_point()
})
```



```
output$plot <- renderCachedPlot({
  ggplot(diamonds, aes(
    carat, price, color = !!input$color_by
  )) +
    geom_point()
}, cacheKeyExpr = {input$color_by})
```

Die Data Science Spezialisten.

eoda GmbH
Universitätsplatz 12
34127 Kassel
www.eoda.de
info@eoda.de
+49 561 202724-40


@eodaGmbH


blog.eoda.de


@eodaGmbH

eodaGmbH