

Reduce, Reuse, R Shiny:

A framework for building generalized and extensible R Shiny applications

Nathan Kosiba
Cytel Inc.
Durham, NC, USA

Agenda

- Introduction
- R Shiny Concepts
- Application Framework
- Code Overview
- Conclusion

Introduction

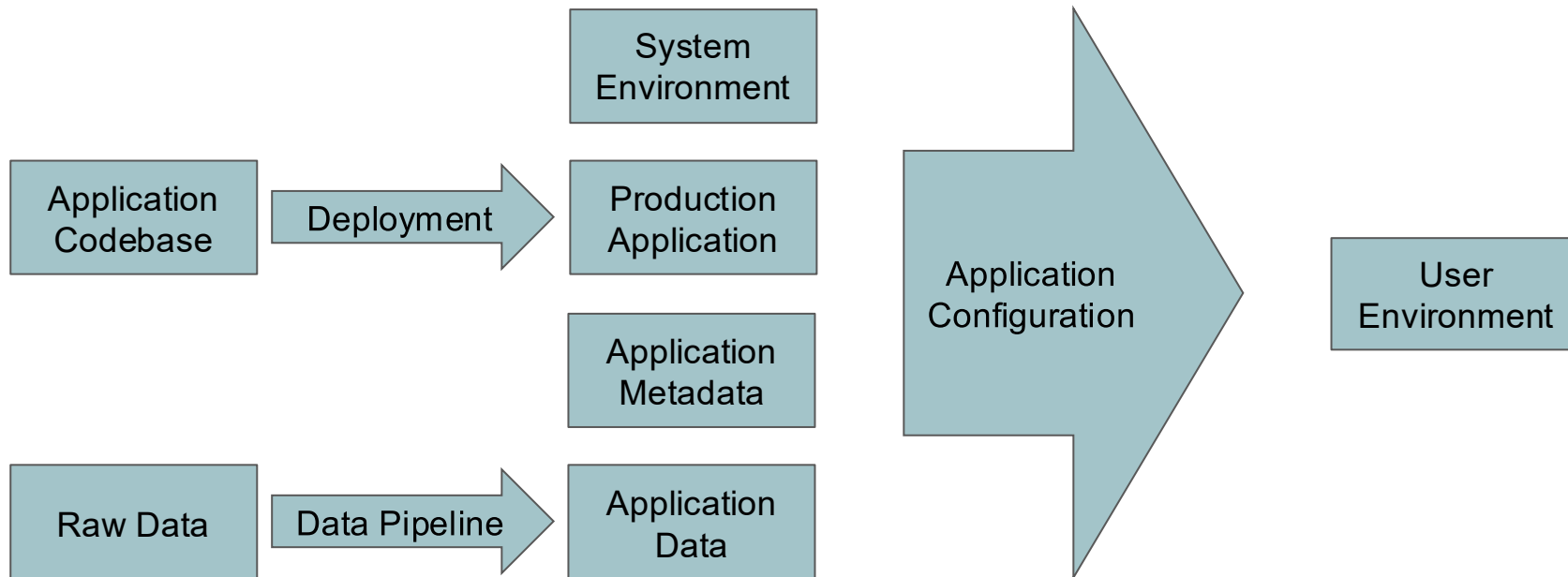
- In my 10th year in industry
- First half traditional statistical programming
- Second half R programing from logging package development to production Shiny applications
- Maintainer of {logrx} and get to work with a great team



R Shiny Concepts

- **System Environment Variables**
 - Configured in local or deployment environments
 - Way to store information for configuration
- **Reactivity**
 - Ability to take a change in an application element and turn it into an action
 - Create a network of interactions within an application
- **Shiny Modules**
 - A paired UI and a Server function, reusable within an application
 - Namespaced by a unique ID

Application framework



Data

Subject Data

- One record per subject
- Useful for app utilities
- Views are about gathering metrics

Event Data

- One record per event
- Care about date and terminology
- Build a patient story and provide context

Results Data

- One record per result
- Trends and Timepoints

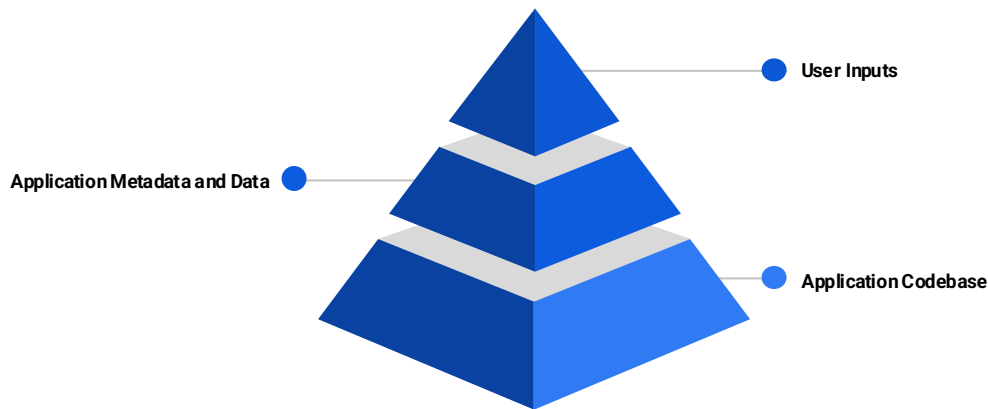
Pipelines

- How data goes from raw to application ready
- Should be easily repeatable and need little to no intervention
- Work backwards, think about application requirements for data



Metadata

- In standard dataset programming, metadata is used to outline the contents and presentation of a variable
- In application programming, we use metadata to outline how to use the data we are given
- Provided through a flat file (JSON, CSV) or database
- Configures code to create views



Extensible Views

- Extensible: designed to allow the addition of new capabilities and functionality
- Allows for clean and concise views that can be configured flexibly
- Extensibility does not mean that it needs to do everything



Extensible Views

Inputs

- Created manually
- Explicit elements
- Choices and options set by developer
- Direct mapping to other elements



- UI Containers
- Choices and options set through metadata
- Mapping will be programmatically generated

Plots and Visuals

- Configured manually by developer
- User plot controls
- Gives all options to users



- Configured through metadata
- Create concise customized views
- Less user options but not less application options

Reactivity

- Explicit referencing
- Generated by static elements
- Basic observers with explicit loops



- One of the biggest challenges to extensible programming
- Referenced through generated IDs
- Observer code may be generated based on elements
- More complicated network of interactivity

UI Code

```
# Traditional UI and server reactive code
```

```
ui <- function(id) {  
  ns <- NS(id)  
  tagList(  
    selectInput(ns("trt_filter"), "Filter Treatment",  
                c("Placebo", "High Dose", "Low Dose")),  
    selectInput(ns("x_axis_var"), "X-Axis Variable",  
                c("sex", "race", "region")),  
    selectInput(ns("y_axis_var"), "Y-Axis Variable",  
                c("age", "height", "weight")),  
    plotOutput(ns("demographics_plot"))  
  )  
}
```

```
# Extensible UI and server reactive code
```

```
ui <- function(id) {  
  ns <- NS(id)  
  tagList(  
    uiOutput(ns("filtering_output")),  
    selectInput(inputId = ns("x_axis_var"),  
                label = "X-Axis Variable",  
                choices = NULL),  
    selectInput(inputId = ns("y_axis_var"),  
                label = "Y-Axis Variable",  
                choices = NULL),  
    plotOutput(ns("demographics_plot"))  
  )  
}
```

Server Basics

```
server <- function(id) {  
  moduleServer(id, function(input, output, session) {  
    demographics_data <- read.csv(file = Sys.getenv("demog_path"))  
  })  
}
```

```
server <- function(id) {  
  moduleServer(id, function(input, output, session) {  
    ns <- session$ns  
    rvs <- reactiveValues(  
      subset_demo = list()  
    )  
  
    # read in demographics data and metadata flexibly  
    demographics_data <- read.csv(file = Sys.getenv("demog_path"))  
    app_meta <- reactive({read_json(file = Sys.getenv("meta_path"))})  
  })  
}
```

Server Code Extensible

```
# turn metadata into selections
output$selections_output <- renderUI({
  for(var in names(app_meta()$demographics$filter_vars)) {
    selectInput(ns(paste0("filter_", var)),
                choices = sort(unique(demographics_data[[var]])))
  }
})

# update inputs using metadata
observe({
  updateSelectInput(session, inputId = "x_axis_var",
                    choices = app_meta$demographics$x_axis_vars)
  updateSelectInput(session, inputId = "y_axis_var",
                    choices = app_meta$demographics$y_axis_vars)
}) %>%
# bind to the app meta reactive object and ignore init and NULL
# this gives us an event that only triggers on application start
bindEvent(app_meta, ignoreNULL = TRUE, ignoreInit = TRUE)

# subset data based on filters
observe({
  temp_demo <- demographics_data
  for(var in names(app_meta()$demographics$filter_vars)) {
    temp_demo <- temp_demo %>%
      filter(var %in% input[[paste0("filter_", var)]])
  }
  rvs$subset_demo <- temp_demo
})
```

Plot Code

```
output$demographics_plot <- ({  
  demographics_data %>%  
    filter(trt %in% input$trt_filter) %>%  
    ggplot(aes(input$x_axis_var, input$y_axis_var)) +  
      geom_boxplot()  
})
```

```
# update output plot  
output$demographics_plot <- ({  
  plot <- ggplot(rvs$subset_demo(),  
                 aes(input$x_axis_var,  
                     input$y_axis_var))  
  # if loop to switch plot based on metadata  
  # this could be a function to be more organized  
  if (app_meta()$demographics$type == "line") {  
    plot +  
      geom_line() +  
      geom_point()  
  } else if (app_meta()$demographics$type == "box") {  
    plot +  
      geom_boxplot()  
  }  
})
```

Conclusion

While starting by building applications entirely using this framework is likely impractical, integrating these concepts and changing your thinking slowly will eventually evolve your workflow to create more extensible applications

These are just building blocks to becoming a better developer

