Co-lab Shiny Workshop

Shiny Server

November 21, 2019

thomas.balmat@duke.edu
rescomputing@duke.edu

In past sessions of the series we developed Shiny applications to implement various analyses using packages such as `ggplot2`, `plotly`, `DT`, and `visNetwork`. Once the necessary `ui()` and `server()` functions were declared, we used `runApp()` to launch an app, which then executed locally on our laptops or on RStudio Cloud. When launched, an app presented a web page in our local browser, based on the Shiny instructions contained in `ui()` and we were able to used features programmed in `server()`. Although convenient for personal use, this method of execution limits access to the host (your laptop) that executed `runApp()`. To publish an app, making it available to the public, we use Shiny Server.

1 Overview

- Preliminaries
  - What can Shiny Server do for you?
  - What are your expectations of this workshop?
  - Interest in R Day?
- Examples
- Resources
- Workshop material
- Using a local TCP/IP port as a server process
- Advantages of Shiny Server over local execution
- Configuring a Shiny Server environment on Linux
  - Log-in to VM
  - Install R
  - Install R packages
  - Install Shiny Server
  - Configure Shiny Server group and workspace
  - Install MySQL (optional)
- Upload and test Shiny scripts and data
  - Upload and test a Shiny script
  - Upload data from git
  - Upload data from Duke Box

5 Using a Local TCP/IP Port as a Server Process

Workshop file `/App/ShinyPort/Port.r`

Given defined `ui()` and `server()` functions, a Shiny app is launched, locally, with a call to `runApp()` such as

```
runApp(list("ui"=ui, "server"=server))
```

Default values for important parameters in this call are (use `?runApp` in your RStudio console to list parameters):

- `port = getOption("shiny.port")`
- `launch.browser = getOption("shiny.launch.browser", interactive())`
- `host = getOption("shiny.host", "127.0.0.1")`

This, typically results in the following call:

```
runApp(list("ui"=ui, "server"=server), port = NULL, launch.browser = TRUE, host = 127.0.0.1)
```

which instructs to launch the app on your local system (127.0.0.1), have the operating system assign an unused TCP/IP port to listen on, and launch the app in your default browser. Executing the above `runApp()` call issues a message such as:

```
Listening on http://127.0.0.1:3997
```

and your R session is locked (the > prompt is no longer available), since your session has become a server-like process listening on the assigned port. Your browser then displays the app as in figure 1. Note the IP address (127.0.0.1) and assigned port (3997) in the browser's url target control.

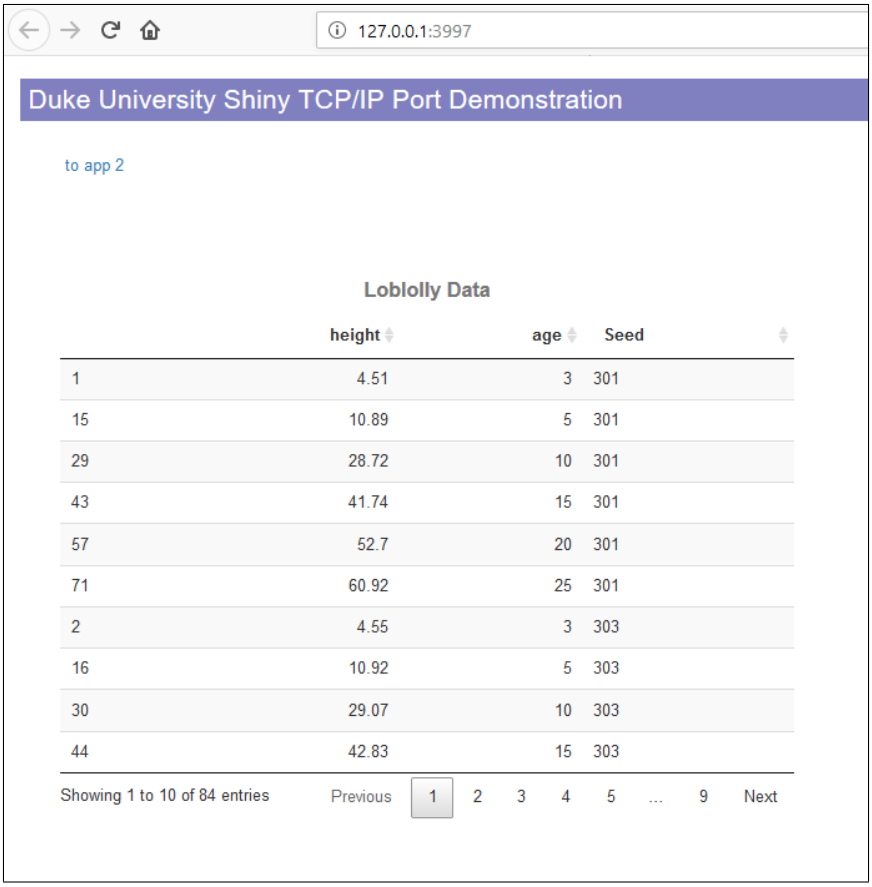

Figure 1: Shiny app displayed in browser as a result of typical `runApp()` call.

At this point, we can launch another, independent, instance of the app by opening a new browser tab and typing the IP address and port that the app is listening on (`127.0.0.1:3997`, in this case).

Suppose that we have two apps and would like to mimic a server environment by having each listen on ports assigned by us so that we can transfer between them using HTML anchor links. Launching the first app with

```
runApp(list("ui"=ui, "server"=server), port=3397, launch.browser=F, host="127.0.0.1")
```

and the second app with

```
runApp(list("ui"=ui, "server"=server), port=3398, launch.browser=F, host="127.0.0.1")
```

will silently launch each app. Ports 3397 and 3398 are assumed to be available and an error message is produced if either are not. We can render the first app in our browser by targeting `127.0.0.1:3397` then, assuming the first app has an anchor reference, such as `<a href=127.0.0.1:3398>anchor_text</a>`, the second app is available by clicking the link.

This is demonstrated in the `/App/ShinyPort/Port.r` script. Executing once with `app <- 1` (line 22) and, in an independent session, with `app <- 2`, makes url targets available for `127.0.0.1:3997` and `127.0.0.1:3998`. Try it. Targeting 127.0.0.1:3997 should produce a browser tab as in figure 1. Clicking the `to app 2` link should produce a tab as in figure 2.

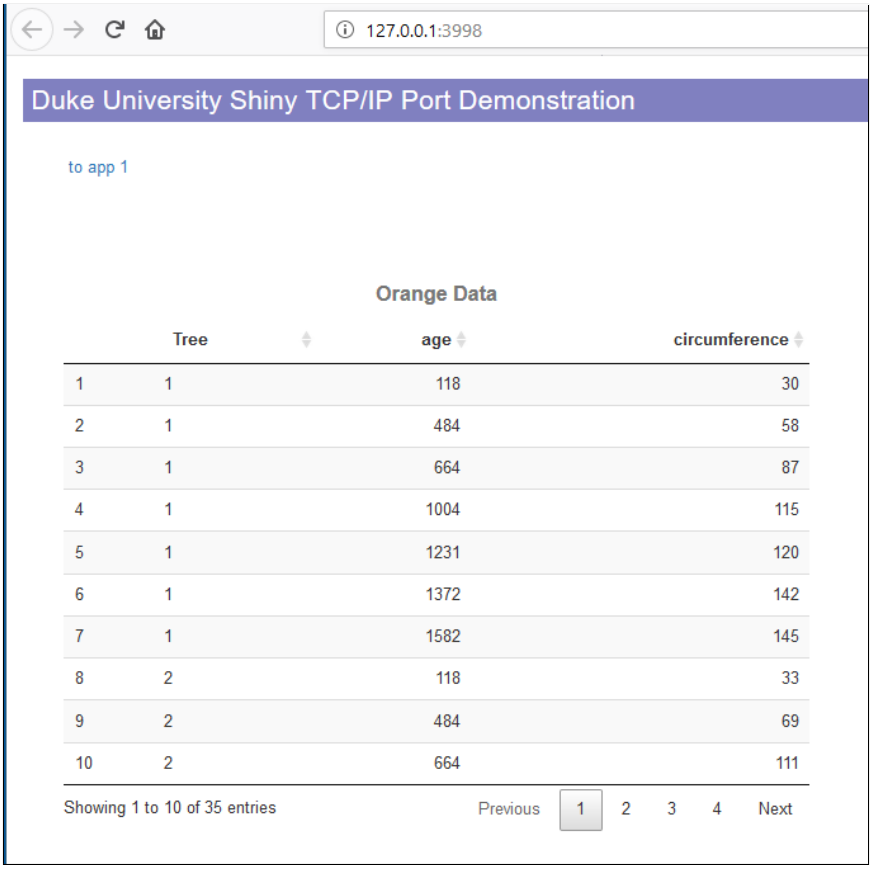

Figure 2: Shiny app displayed in browser as a result of targetting a url with assigned port in `runApp()`.

6 Advantages of Shiny Server Over Local Execution

Shiny Server is a server process, which offers the following advantages over local execution:

- Your app (web site) is referenced using a public url (a google bot will list you, provided they are permitted access by governing firewalls)
- Twenty-four hour public access (or scheduled according to a hosting plan)
- File and data access privileges as supported by the host operating system
- Global memory sharing between active apps (time-out regulated)
- Package version and feature stability, due to "everyone" executing a single version of an app
- Improved logging of web server and application messages
- Ability to "track" usage by origin ip address and request-specific context
- Increased access to resources, since your server is likely hosted by an outstanding team within your local OIT

An important consideration is that, generally, locally developed apps require little modification to properly function in a Shiny Server environment. The most common modification is to database server paths and file locations, since these are, customarily, also migrated to a server, possibly the same one that hosts your Shiny apps.

7 Configuring a Shiny Server environment on Linux

7.1 Log-in to VM

A VM from Research Toolkits (https://rtoolkits.web.duke.edu) will be assigned to you. Connect to it using ssh. Log-in with your Duke netID. Your initial password is your netID. You will be prompted to change it on first log-in. Note that multi-factor authentication is not performed. All VMs will be retired within one week of this Shiny Server Co-lab session. Contact me (thomas.balmat@duke.edu) if your VM is not available during this period. Note that the VMs are accessible only while connected to Duke's network. A VPN connection satisfies this requirement. An alias can be specified for your VM that will form your Shiny Server url. We will define these once all server components and an application have been configured.

7.2 Install R

Enter super-user mode

```
# sudo -i
```

Update Ubuntu package list

```
# apt-get update
```

Install R

```
# apt-get install r-base
```

Test your R installation

```
# R
```

7.3 Install R Packages

From the R prompt:

```
> install.packages("shiny")
> install.packages("ggplot2")
> install.packages("DT")
> install.packages("rmarkdown")
> install.packages("plotly")
> install.packages("visNetwork")
```

Test:

```
> library(ggplot2)
> g <- ggplot() + geom_point(aes(x=1:100, y=1:100))
> str(g)
```

Quit R:

```
> q()
```

7.4 Install Shiny Server

```
# apt-get install gdebi-core
# wget https://download3.rstudio.org/ubuntu-14.04/x86_64/shiny-server-1.5.9.923-amd64.deb
```

```
# gdebi shiny-server-1.5.9.923-amd64.deb
```

This should result in the following message (with the current date and time and your server ID):

```
# Nov 20 21:28:20 rapid-1148.vm.duke.edu systemd[1]:  Started ShinyServer.
```

Test Shiny Server status (I included the sudo prefix in case you execute this later):

```
# sudo systemctl status shiny-server
```

Check for the green active message:

```
shiny-server.service - ShinyServer Loaded:  loaded (/etc/systemd/system/shiny-server.service;
enabled; vendor preset:  enabled)
Active:  active (running) since Wed 2019-11-20 21:28:20 EST; 7ms ago
```

Stop/start/restart Shiny Server:

```
# sudo systemctl stop shiny-server
# sudo systemctl start shiny-server
# sudo systemctl restart shiny-server
```

Test the web server by entering the Shiny Server url, http://rapid-1148.vm.duke.edu:3838/ (replace `rapid-1148` with your VM ID), into your browser's url field. The result should be similar to that in figure 3.

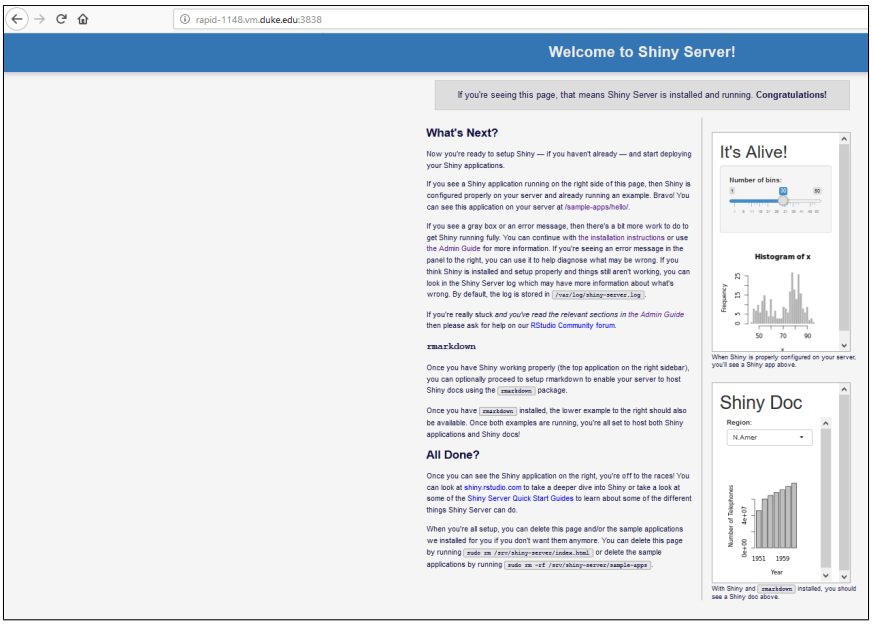

Figure 3: Shiny Server default. Executed by visiting `http://rapid-1148.vm.duke.edu:3838/`

Note the use of Shiny Server's default TCP/IP port 3838. This is taken from the `listen` entry of `/etc/shiny-server/shiny-server.conf`, which appears below.

```
# Instruct Shiny Server to run applications as the user "shiny"
```

```
run_as shiny;
# Define a server that listens on port 3838
server {
  listen 3838;
  # Define a location at the base URL
  location / {
    # Host the directory of Shiny Apps stored in this directory
    site_dir /srv/shiny-server;
    # Log all Shiny output to files in this directory
    log_dir /var/log/shiny-server;
    # When a user visits the base URL rather than a particular application,
    # an index of the applications available in this directory will be shown.
    directory_index on;
  }
}
```

7.5 Configure Shiny Server Group and Workspace

Add yourself to the shiny group (that was created during Shiny installation):

```
# sudo usermod -aG shiny tjb48
```

Disable directory searches:

- `# vi /etc/shiny-server/shiny-server.conf`
- Change the `directory_index` entry from `on` to `off`
- Save the file.

Create an app directory and file:

- `# cd /srv/shiny-server/`
- `# mkdir app`
- `# chown shiny:shiny app`
- `# cd app`
- Copy contents of workshop file `App/NPDHist/app.R` to clipboard and paste into `/srv/shiny-server/app/app.R` on the server
  - `# cat > app.R`
- Copy contents of workshop file `App/NPDHist/style.css` to clipboard and paste into `/srv/shiny-server/app/style.cs` on the server
  - `# cat > style.css`

Test the new app by visiting http://rapid-1148.vm.duke.edu:3838/app/ (replacing `rapid-1148` with your VM ID). The result should be similar to that in figure 4.

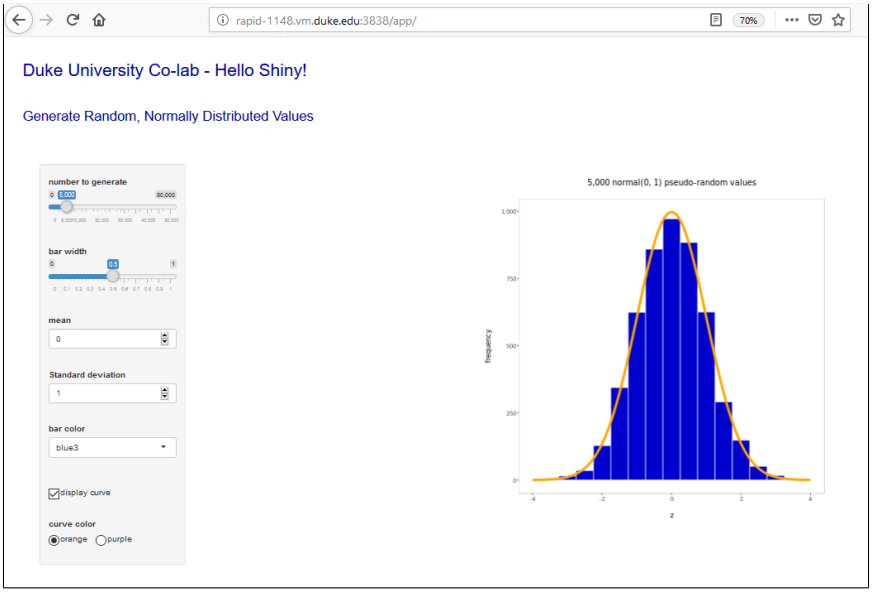

Figure 4: First Shiny Server app. Random normal values.

Replace `index.html` (it is a link, try `ls -al`) to reference the new app:

- `# rm /srv/shiny-server/index.html`
- Copy contents of workshop file `App/index.html` to clipboard
- `# cat > /srv/shiny-server/index.html`

Test by visiting `http://rapid-1148.vm.duke.edu:3838` (replace VM ID). Note that this url targets the top level Shiny Server directory. The new app is now the default app. Are the apps in the `sample-apps` directory still executable?

Delete the sample apps:

```
# rm -r /srv/shiny-server/sample-apps
```

Change the owner of all Shiny Server directories and files:

```
# chown -R shiny:shiny /srv/shiny-server/*
```

7.6 Install MySQL (optional

Install Linux packages:

```
# sudo apt install mysql-server
# sudo systemctl stop mysql.service
# sudo systemctl start mysql.service
# sudo systemctl restart mysql.service
```

Create a database and users:

```
# sudo mysql -u root
# mysql> create user 'tjb48'@'localhost' identified by 'colab';
# grant all on *.* to 'tjb48'@'localhost' with grant option;
# mysql> create database colab;
# mysql> create user 'colabreader'@'localhost' identified by 'colabread';
```

```
# mysql> grant select on colab.* to 'colabreader'@'localhost';
# mysql> quit
```

Install Linux packages required for database connections:

```
# sudo apt-get install libmariadb-client-lgpl-dev
# sudo apt-get install libmysqlclient-dev
# apt-get install libcurl4-openssl-dev
# apt-get install libssl-dev
```

Install R packages (from within R):

```
> install.packages("DBI")
> install.packages("RMySQL", dependencies=TRUE)
> library(DBI)
> library(RMySQL)
```

Test database connection from R:

```
> db <- dbConnect(MySQL(), host="127.0.0.1", dbname="colab", user="tjb48", password="colab")
> dbGetQuery(db, "show tables")
> dbDisconnect(db)
```

8 Upload and Test Shiny Scripts and Data

8.1 Upload and Test a Shiny Script

Use one of yours or one from a previous session of the series.

8.2 Upload Data from git

```
wget https://raw.githubusercontent.com/tbalmat/Duke-Co-lab/master/Session-5/Data/GWASResults.csv
```

8.3 Upload Data from Duke Box

- On the Box site, locate the file, and use the Share button
- Enable Share Link (slider)
- Change "Invited People" to "People with Link" (note that this permits anyone with link to access file; remaining options, such as "People in your company" causes HTML to be returned by wget - likely credentials prompt; remove share or set perms to "Invited only" after download))
- Click "Link settings" and copy the Direct link
- In Linux session use: wget [link from above] -O outfile
  ex: wget https://duke.box.com/shared/static/z0f7l1osc8j5vas4uwhihdd6kp4udihs.txt -O boxfile.txt
- Disable Share Link (slider) in Box site

8.4 Upload Data Using SFTP (optional)

- Examine `/etc/ssh/sshd_config`
- Add to the end of sshd_config
  - `Subsystem sftp internal-sftp`
  - `Match group sftpaccess`
  - `ChrootDirectory %h`
  - `X11Forwarding no`
  - `AllowTcpForwarding no`

- `ForceCommand internal-sftp`

- Restart sshd service
  - `sudo systemctl restart ssh`

8.5 Import SQL Records Using R (optional)

Upload data:

```
# gwas <- read.table("GWASResults.csv", header=T, sep=",", strip.white=T)
```

From within R:

```
> library(DBI)
> library(RMySQL)
> db <- dbConnect(MySQL(), host="127.0.0.1", dbname="colab", user="tjb48", password="colab")
> dbGetQuery(db, paste(" create table gwas(gwasID tinyint, phenotype varchar(100), snp varchar(25), p r
                       " primary key(gwasID, phenotype, snp))", sep=""))
> dbGetQuery(db, "show tables")
> dbGetQuery(db, "grant select on gwas to colabreader@localhost")
> gwas <- read.table("GWASResults.csv", header=T, sep=",", strip.white=T)
> sql <- apply(as.matrix(1:nrow(gwas)), 1,
              function(i) paste(gwas[i,"GWAS"], ", '", gwas[i,"phenotype"], "', '",
                                gwas[i, "SNP"], "', ", gwas[i,"p"], sep=""))
> sql <- paste("insert into gwas values(", paste(sql, collapse="), (", sep=""), ")", sep="")
> dbGetQuery(db, sql)
> dbGetQuery(db, "select * from gwas limit 10")
> dbDisconnect(db)}
```

8.6 Log SQL Queries (optional)

Create a log table:

```
> dbGetQuery(db, "create table queryLog(date smalldatetime not null, query varchar(100) not null)")
> dbGetQuery(db, "create index INQueryLogDate on queryLog(date)")
> dbGetQuery(db, "create index INQueryLogQuery on queryLog(query)")
> dbGetQuery(db, "grant select on queryLog to colabreader")
```

Prior to executing a query, insert a log record:

```
> dbGetQuery(db, "insert into quereyLog values (curr_date(), 'Highly informative GWAS query 1')")
> x <- dbGetQuery(db, "select * from gwas where p<1e-6")
```

9 A Few Points

- http vs. https
- Global memory sharing between apps
- Create an alias for your server (`http://your-alias.rc.duke.edu:3838`)

10 Debugging

It is important that you have a means of communicating with your app during execution. Unlike a typical R script, that can be executed one line at a time, with interactive review of variables, once a Shiny script launches, it executes without the console prompt. Upon termination, some global variables may be available for examination, but you may not have reliable information on when they were last updated. Error and warning messages are displayed in the console (and the terminal session when executed in a shell) and, fortunately, so are the results of `print()` and `cat()`. When executed in RStudio, Shiny offers sophisticated debugger features (more info at https://shiny.rstudio.com/articles/debugging.html). However, one of the simplest methods of communicating with your app during execution is to use `print()` (for a formatted or multi-element object, such as a data frame) or `cat(, file=stderr())` for "small" objects. The `file=stderr()` causes displayed items to appear in red. Output may also be written to an error log, depending on your OS. Considerations include

- Shiny reports line numbers in error messages relative to the related function (`ui()` or `server()`) and, although not always exact, reported lines are usually in the proximity of the one which was executed at the time of error
- `cat("your message here")` displays in RStudio console (generally, consider Shiny Server)
- `cat("your message here", file=stderr())` is treated as an error (red in console, logged by OS)
- Messages appear in RStudio console when Shiny app launched from within RStudio
- Messages appear in terminal window when Shiny app launched with the `rscript` command in shell
- There exists a "showcase" mode (`runApp(display.mode="showcase")`) that is intended to highlight each line of your script as it is executing
- The reactivity log may be helpful in debugging reactive sequencing issues (`options(shiny.reactlog=T)`, https://shiny.rstudio.com/reference/shiny/0.14/showReactLog.html It may be helpful to initially format an apps appearance with an empty `server()` function, then include executable statements once the screen objects are available and configured
- Although not strictly related to debugging, the use of `gc()` to clear defunct memory (from R's recycling) may reduce total memory in use at a given time

11 Querying the Shiny Server Log for Usage Statistics

- Log file record extraction
- Review H2P2 web server and database usage map