Supplement to Shiny in Production

2019-01-05

Contents

Chapter 1

Shiny in Production Workshop @ RStudio Conf 2019

This document is full of supplemental resources and content from the Shiny in Production Workshop delievered at rstudio::conf 2019.

The **bookdown** package can be installed from CRAN or Github:

```
install.packages("bookdown")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Chapter 2

Introduction to Shiny in Production

2.1 Why are we here?

> Shiny applications are being deployed in high-value, customer-facing, and/or enterprise-wide scenarios. Unfortunately, they are often being done without the benefit of best practices. This workshop will help you and/or your IT colleagues who support your data scientists learn how to accelerate a successful Shiny application deployment in production scenarios.

Over the past year, software developers at RStudio have been working hard to dispel rumors that Shiny "isn't ready and can't run in production". They've built a bunch of cool new tools that are useful in preparing applications for production and understanding how to configure and scale them for optimized performance and user experience.

This workshop will cover all these new tools for shiny development as well as the equally important logistical pieces of a production story:

- What does production infrastructure and tooling look like for Shiny apps?
- How do we get Shiny apps from development into production?
- How are Shiny apps maintained production?

> When developers begin to think of infrastructure as part of their application, stability and performance become normative. - Jeff Geerling "Ansible for DevOps"

2.1.1 Can Shiny be used in production?

2.1.2 Objectives

2.1.2.1 Understand the importance of incremental changes and testing

- Version control
- Tests for package upgrades
- Use of separate environments for staging and production
- Incorporating automated testing into a development workflow: `shinytest`

2.1.2.2 Data product tradeoffs

- What are the advantages to using Shiny vs Plumber vs R Markdown
- What is the difference between a stateless Plumber API and a Shiny Session?

2.1.2.3 Development vs. Production environment considerations

- Defining a data model
  - Working with databases
- Tools for understanding application performance
  - `shinyloadtest`
  - `profvis`
- Tools for improving application performance
  - Plot caching
  - Synchronous vs asynchronous paradigms: `async`

2.1.2.4 Deployment architecture and tools

- Introduction to analytic infrastructure
- Configuration management
- Resources for scaling horizontally

2.2 Outline

2.3 Workshop Infrastructure

- RStudio Connect
- PostgreSQL

Instructions for accessing the classroom environment are available in the workshop slide deck.

Chapter 3

Introduction to the Application

3.1 Every Application has an Origin Story

Data Scientists at RStudio University have discovered that there are trackable traits and behaviors students engage in that have been predictive of the desired 4-year graduation track.

They have built a shiny application that can be used by the very data-savvy advisors at this illustrious institution to identify students in need of guidance and show them the top behavioral factors driving individual predictions coming out of the model.

The POC was a smashing success - but now *the advisors actually want to use this thing for real.*

- We've developed a nice app
- We want to put it into production
- We want confidence that it will perform well in production, both now and in the future

3.1.1 Activity: Explore the Application

Open the POC Application Run the Application Explore the Application code

1. Are there any parts of the application code that don't make sense?
2. Brainstorm: what qualifies as production?
3. Brainstorm: 5 things to consider when bringing this application into production.

What are our application requirements? - Who does this app serve? - What kind of utilization do we expect? - Expected concurrent usage, can we handle peaks/spikes? - What happens if we under budget? Do we have strategies for scaling up? - What happens if we over budget? Do we have strategies for scaling down? - How will this all be monitored?

Discussion

- Is this app ready for production?
- What insights would be useful to have before taking this app into production?
- What tools currently exist that would help us run tests to gain these insights?

Create a checklist for taking this (any?) application into production

- What is your current process for taking applications into production?

3.1.2 Is Shiny the right medium (data produt) for this project?

- Alternative Architectures (later chapter)

3.2 Understanding the App: `reactlog`

```
library(shinyreactlog)
options(shiny.reactlog = True)
runApp()
```

React Log Visualizer Reference

> For security and performance reasons, do not enable shiny.reactlog in production environments. When the option is enabled, it's possible for any user of your app to see at least some of the source code of your reactive expressions and observers.

3.3 Checklist for Taking Applications into Production

A high-level Checklist to build off of:

- [ ] Tests
- [ ] Performance Optimization
- [ ] Environment (Package) Management
- [ ] Data Access
- [ ] Deployment Hand Off
- [ ] Scaling
- [ ] Monitoring

Chapter 4

Application Testing: `shinytest`

[From the Webinar] - You've developed a nice app - You've put it in production - You want to be confident that it will keep running in the future

Things that can change/break a Shiny application - Modifying code - Upgrading the `shiny` package - Upgrading other packages - Upgrading R - External data source changes or fails

4.0.1 Testing Options

- Manual testing
  - time intensive
  - inconsistent
- Automated testing (hard)
  - web browser
  - simulated user interactions
  - tests for graphical elements

Shinytest: https://github.com/rstudio/webinars/blob/master/48-shinytest/shinytest.pdf

[demo on the Geyser app]

4.1 From the Blog

Blog

`shinytest` is a package (available on CRAN) to perform automated testing for Shiny apps.

- Record Shiny tests
- Run and troubleshoot Shiny tests

Support for `shinytest` is available in RStudio v1.2 preview

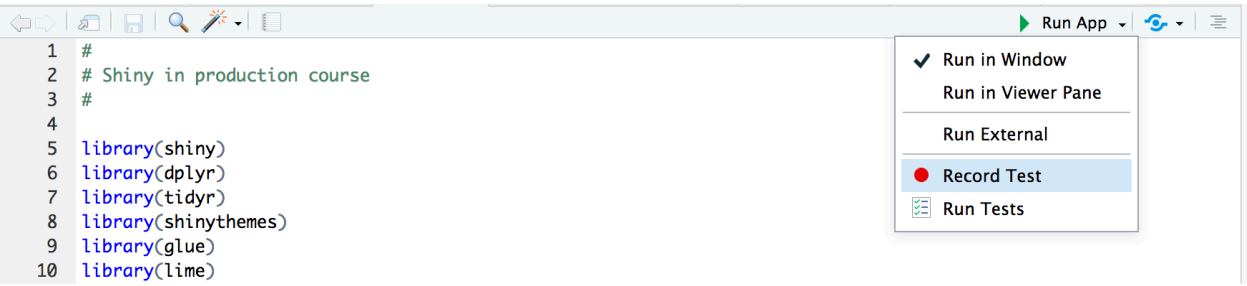

Figure 4.1: Record Test Button

4.2 Installation

'install.packages("shinytest")

Note: When running `shinytest` for the first time, you may be prompted by the RStudio IDE or package warning messages to install some dependencies. `shinytest` requires a headless web browser (PhantomJS) to record and run tests. - To install it, run shinytest::installDependencies() - If it is installed, make sure the phantomjs executable can be found via the PATH variable.

4.3 Record Tests

- Run `recordTest()` to launch the app in a test recorder.
- Create the tests by interacting with the application - this will allow the recorder to snapshot the application state at various points.
- Quit the test recorder. This action will trigger the following events:
  - The test script will be saved as a .R file in a subdirectory of the application named `tests/`.
  - If you are running in the RStudio IDE, it will automatically open this file in the editor.
  - The test script will be run, and the snapshots will be saved in a subdirectory of the `tests/` directory.

To record tests from `R`, run the following:

```
library(shinytest)

recordTest("path/to/the/app")   #Replace with the correct path
```

To record tests from RStudio v1.2, when an application file (app.R, server.R, ui.R or global.R) is open in the editor, a button labeled *Run App* will appear at the top of the editor pane. Click on the small black triangle next to this button to reveal the menu of extended options.

This launches the Shiny application to be tested in a separate R process. We'll refer to this as the **target app**. At the same time, the current R process lauches a special Shiny application which displays the target app in an iframe along with some controls. We'll refer to this as the **recorder app**. You should see something like this:

The panel on the right displays some controls for the test recorder, as well as a list of **recorded events**. As you interact with the target app, you will see those interactions appear in the recorded events list.

For testing a Shiny application, interacting with the inputs is only one part of the equation. It's also necessary to check that the application produces the correct outputs. This is accomplished by taking **snapshots** of the application's state.

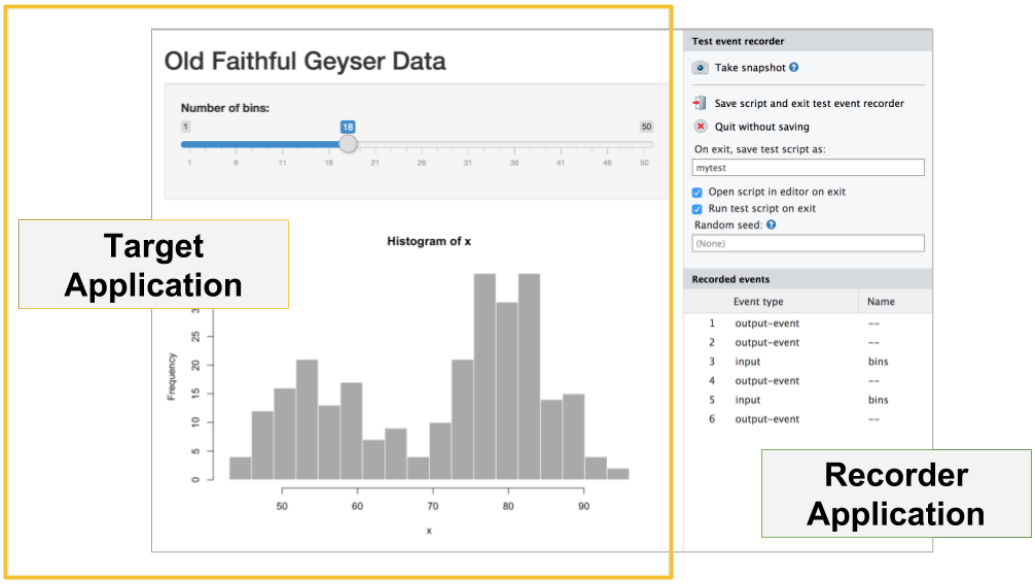

Figure 4.2: Target and Recorder App iframe

To take a snapshot of the application's state, click the *Take snapshot* button on the recorder app. This will record all input values, output values, and exported values.

4.4 Running Tests

When you quit the test recorder, it will automatically run the test script. There are three separate components involved in running tests:

1. First is the **test driver**. This is the R process that coordiates the testing and controls the web browser. When working on creating tests interactively, this is the R process that you use.
2. Next is the **Shiny process**, also known as the **server**. This is the R process that runs the target Shiny application.
3. Finally, there is the **web browser**, also known as the **client**, which connects to the server. This is a headless web browser - one which renders the web page internally, but doesn't display the content to the screen (PhantomJS).

So, when you exit the test recorder, it will by default automatically run the test script and print something like this:

```
Saved test code to /path/to/app/tests/mytest.R
Running mytest.R
====== Comparing mytest ...
  No existing snapshots at mytest-expected/. This is a first run of tests.

Updating baseline snapshot at tests/mytest-expected
Renaming tests/mytest-current
      => tests/mytest-expected.
```

Figure 4.3: View Failed Tests

This is the result of running `testApp()`, which can also be manually run by providing the desired application and test like this:

```
testApp("exampleApp", "mytest")
```

The built-in integreation with RStudio v1.2 provides `Run Tests` as a drop down menu option in your Shiny app source file (see it located under the *Record Test* option in the screenshot above).

4.5 Subsequent Test Runs

After the initial test run, you can continue to run the tests to check for changes in application behavior.

If there are any differences between current and expected results, the test output will look something like this:

```
Running mytest.R
====== Comparing mytest ...
  Differences detected between mytest-current/ and mytest-expected/:

    Name         Status
    001.json  != Files differ
    001.png   != Files differ
Would you like to view the differences between expected and current results [y/n]?
```

To view failed tests in the RStudio IDE, go to the *Build tab* and make sure the *issues toggle* is selected:

For each test with different results, you can see the differences between the expected and current results.

4.6 Testing Code

The `shinytest` package was created for testing Shiny applications on the interaction-level. To test Shiny code and functions, we suggest using the `testthat` package.

Info GitHub

Learn about testing and how to setup test workflow and structure: R packages by Hadley Wickham

Chapter 5

Profiling: “The most important thing”

5.1 `profivs`

Webinar Slides

Chapter 6

Deployment: RStudio Connect

6.1 RStudio Connect

6.2 Packrat

Chapter 7

Connecting to Data in Production

7.1 Databases

7.2 `config`

Chapter 8

Load Testing

8.1 `shinyloadtest`

Webinar Slides

Chapter 9

Plot Caching

Chapter 10

Scaling

10.1 Application Scaling 101

- cooking/ kitchen metaphor

10.2 RStudio Connect Performance Settings

Chapter 11

DevOps Philosophy & Tooling

11.1 DevOps Vocab 101

11.2 Design and Management of Analytic Infrastructure (alternative architectures)

11.3 Programatic Deployment into RStudio Connect

Chapter 12

Production Case Studies

12.1 Case Study A: Dev/Test/Prod

12.2 Case Study B: CI, Git, Chef

12.3 Case Study C: Docker

Chapter 13

Alternatives to Shiny

13.1 Plumber

13.2 R Markdown

Chapter 14

Shiny Async

Webinar Slides