PharmaSUG 2021 - Paper AD-205

Productionalizing Shiny Deployments

Phil Bowsher, RStudio PBC;
Sean Lopp RStudio PBC;
Kelly O'Briant RStudio PBC;
Cole Arendt RStudio PBC

Abstract

Shiny is a powerful tool in the pharmaceutical space as it can help modernize many legacy processes and reporting workflows. This paper will highlight many new tools that can help harden (productionalize) shiny apps. The audience for this paper are those who have wondered how to approach production Shiny deployments. Using Shiny for submissions is a related topic but will not be the focus of this paper. The R in Pharma gathering has had related talks like Joe Cheng's keynote here:

https://speakerdeck.com/jcheng5/using-shiny-responsibly-in-pharma

The examples for this paper are housed at the link below:

https://github.com/philbowsher/Advanced-Shiny-Deployments

Introduction

For many, it is a happy accident when a Shiny app becomes popular. As the consumption increases, corporate IT often requires that additional steps be taken to put the app into production. Often these actions require knowledge of various software development/engineering tools that are well established but new to many statistical programmers. This paper will provide an introduction to some of these topics and tools for the pharma statistical programming community. This paper will not be an intro into some tools or topics but resources will be provided.

Mental Model for Shiny in Production

You have a successful app for production: Great Job! Below is a mental model of three high level, logistical pieces/framework of your production story:

What does production infrastructure and tooling look like for Shiny apps?

- Package dependency management
- Add/Use CI/CD

How do we get Shiny apps from development into production?

- Testing
- Deployment to Production with a version control system

How are Shiny apps maintained in production?

- Manage data via ETL (extract, transform, load)
- Modularizing the app

Tooling

Below are a list of tools/topics that are important in this space, some of which are out of scope for this paper but worth investigating:

- Package Management
- Docker (Not covered in this paper)
- R Markdown (ETL Job)
- Pins
- Shiny modules (Not covered in this paper)
- renv
- Shinytest
- Git/Github
- YAML
- Github Actions (CI/CD)

Agile

It is important to have a system for iterating quickly and this is a key part of agile software engineering. Fast feedback is an important principle in DevOps (Development Operations / app deployment) and you can read more about it here:

https://blog.axosoft.com/feedback-loops-agile-development/

It is important to have a production environment like RStudio Connect as a platform for doing the "sharing" and "optimization" tests as reviewed below. So for data scientists and clinical statistical programmers it is important to:

- Share an R notebook before you invest in building a dashboard
- Share a dashboard before building a Shiny app prototype
- Share an app prototype before you invest in optimization
- Optimize an R/python app before you invest in a team of software engineers to harden/translate it

Production

There are three core pieces of analytic infrastructure that every Shiny developer needs to be successful:

1. An IDE (integrated development environment) capable of supporting work in the R ecosystem.

2. A production-like environment; capable of hosting applications built in R, and easily accessed for iterative testing.

3. Access to R packages and R environment restoration tools.

Organizations that invest in making this infrastructure a priority are the most successful in productionalizing work produced in R (and Shiny). When developers do not have components of this system in place, anti-patterns emerge. Having a capable development environment is table stakes for any Shiny development work to begin. Agile methodology necessitates access to production environments like RStudio Connect which can ensure that the code, results, and dependencies are isolated, easily accessible, and safely locked away. We recommend that your finished project code, results, and artifacts be deployed outside of the RStudio development IDE as a final step. Moving these final results into a secure, stable, reproducible, and easily shared archive such as RStudio Connect is like locking a copy in a safety deposit box.

Joe Cheng gave a wonderful keynote presentation on productionalizing Shiny at the RStudio Conference in 2019. The talk highlights many topics such as shinytest, shinyloadtest, profvis, plot caching and async. This paper plans to build off of that talk but not to replicate ideas first presented there. You can watch the video here:

https://www.youtube.com/watch?v=Wy3TY0gOmJw&ab_channel=RStudio

Books

This paper will go through an example of productionalizing a Shiny deployment. We will not discuss how to create a shiny application. The following new book (free online) is a wonderful resource if you are new to this space:

https://mastering-shiny.org/

Much of the content of this paper has been adopted from previous efforts by Kelly O'Briant and you can read her book here:

https://kellobri.github.io/shiny-prod-book/

In this paper, we will discuss in detail Git and Github but we will not provide an introduction. If you are new to Git, please see this wonderful book by Jenny Bryan:

https://happygitwithr.com/

R Package Strategy

One of the most important aspects of Shiny in production is having a package strategy that is understood by developers. You can read more about this here:

https://environments.rstudio.com/reproduce.html

Key aspects are:

- Aligning on supported R versions
- Not upgrading R (instead, install multiple versions)
- Not installing packages from CRAN but using IT owned internal repositories
- Aligning historic R versions to frozen package snapshots

Pins

Pins is an R package that allows scientists to place resources (e.g. CSV file) locally or in remote storage. This allows Shiny developers to "point" to resources rather than save files in their app folders.

https://pins.rstudio.com

For example, imagine you have a Shiny app that loads some data like this:

```
dmae <- read_sas("dmae.sas7bdat") %>% mutate(
  SEX = factor(SEX),
  AESEV =  factor(AESEV),
  RACE = factor(RACE)
)
```

Using, Pins, one could deploy that artifact to a central location like this:

```
# Publish Pin
```{r}
library(pins)

pins::board_register_rsconnect()

board_register("rsconnect",
        server = "https://rstudio-connect-server",
        key = "PutYourKeyHere")

pin(dmae, name = "DMAE" , description = "Adverse Events", board = "rsconnect")
```
```

Then you can retrieve the Pin in the Shiny app with this code:

```
```{r}
library(pins)

dmae <- pin_get("DMAE", board = "rsconnect")
```
```

The important take-away here is that the Pin-definition code above could be put into a script or R Markdown file and then scheduled to run automatically as a lightweight ETL job. Tools like cron and RStudio Connect can make it possible to run these ETL jobs on defined intervals, and the output can be a pinned dataset that is fed into the production Shiny app. This will separate out the heavy data wrangling from the app.

API Keys allow you to connect different tools and are an important part of the software development space. It is likely users will need to manage various API keys for connecting tools. You will see examples of this below with Github Actions. Using Pins with RStudio Connect also requires an API key as in the code above. The code above is a simple example but it is advised to place your API token as an environment variable in your `.Renviron` file. The usethis package makes it easy to edit this file for this purpose:

https://usethis.r-lib.org/

The code would look like this:

usethis::edit_r_environ()

Then enter in the file and save:

CONNECT_API_KEY="your-api-key"

CONNECT_SERVER="https://rstudio-connect-server"

These API keys allow users to programmatically access content on RStudio Connect and use the Connect Server API. We will revisit API keys below in the Github Actions section. You can read more about using API keys with RStudio Connect here:

https://docs.rstudio.com/connect/user/api-keys/

RENV

renv is a R package that helps to create reproducible environments for your R projects including shiny apps. This ensures reproducibility by having project-local dependencies stored in a project library for your work.

https://rstudio.github.io/renv/

1. **renv::init()** - initialize a new project-local environment with a private R library
2. Work in the project as normal, installing and removing new R packages
3. **renv::snapshot()** - save the state of the project library to the lockfile (called renv.lock)
4. Work in the project as normal, installing and removing new R packages
5. **renv::restore()** - revert to the previous state as encoded in the lockfile

If for some reason, a Shiny programmer needs to do development in a new environment, or if a colleague needs to replicate the environment that was used previously, this can be accomplished with: **renv::restore()**

SHINYTEST

shinytest provides a simulation of a Shiny app that you can control in order to automate testing. There are many reasons why you might want to test your apps:

- Automatically identify if an upgraded R package has different behavior. (This could include Shiny itself!)
- Test optimizations to your application code to see if it changes the application behavior,
- Automatically detect if an external data source stops working, or returns data in a changed format.

The general workflow for creating a test is:

1. Run **recordTest()** to launch the app in a test recorder.
2. Create the tests by interacting with the application and telling the recorder to snapshot the state at various points.
3. Quit the test recorder.

Many pharmaceutical companies require QA (Quality Assurance) to run tests on code used to generate reports. These tests can be done with shinytest. You can see some sample shinytest code here:

https://github.com/colearendt/shinytest-playground/tree/main/tests

The general workflow for running a test is:

1. Run tests with **shinytest::testApp()**
2. If there are differences, view the differences with **shinytest::viewTestDiff(testnames = "mytest")** (for the mytest test)

We will revisit shinytest with CI/CD (continuous integration and continuous deployment).

GIT/GITHUB

Git is a powerful tool to use for productionalizing shiny apps. Git is an example of a general software development tool that is starting to be a big part of clinical development. Git allows for a team of Shiny developers to collaborate and work on one code base. An introduction to Git is outside the scope of this paper, but the book listed above is a great resource. Below we will discuss how to deploy a Shiny app from Git using RStudio Connect. A similar process would be used for other production environments.

RStudio Connect supports a variety of deployment methods that integrate with Git. One option is to use a built in **polling** mechanism as described here:

https://docs.rstudio.com/connect/user/git-backed/

1. Create a manifest.json in the Shiny app repository using **rsconnect::writeManifest()**. This is a function in the rsconnect R package: https://rstudio.github.io/rsconnect/
2. Setup RStudio Connect to poll the Git repository for changes.
3. Commit your code and manifest to a Git repository.
4. RStudio Connect will watch the repository and deploy any new commits that it detects.

This workflow is sufficient for most projects. Alternatively, you can use a CI/CD system to implement a **push** mechanism. In this mode, you would commit changes to a repository, and then a CI/CD system would push those changes to RStudio Connect using deployment APIs available on Connect. CI/CD systems can be used to push updates to other production systems if you are not using RStudio Connect.

YAML

YAML is a human-readable data-serialization language that is used often to instruct files how to behave. R Markdown even uses YAML in the top of its files where users specify the title, date etc. YAML is commonly used for configuration files and in applications where data is being stored or transmitted. YAML files are becoming standard as the method for directing and guiding your workflows. For example, YAML is used below with Github Actions to automate our workflows. Below is a nice resource if learning YAML is needed:

https://geekflare.com/yaml-introduction/

The R Markdown YAML metadata tells R Markdown, pandoc, and other software exactly how to handle the document. ymlthis is a package to help you write YAML metadata for R Markdown documents and related tools like blogdown, bookdown, and pkgdown. The below resource provides articles for people new to YAML in R Markdown:

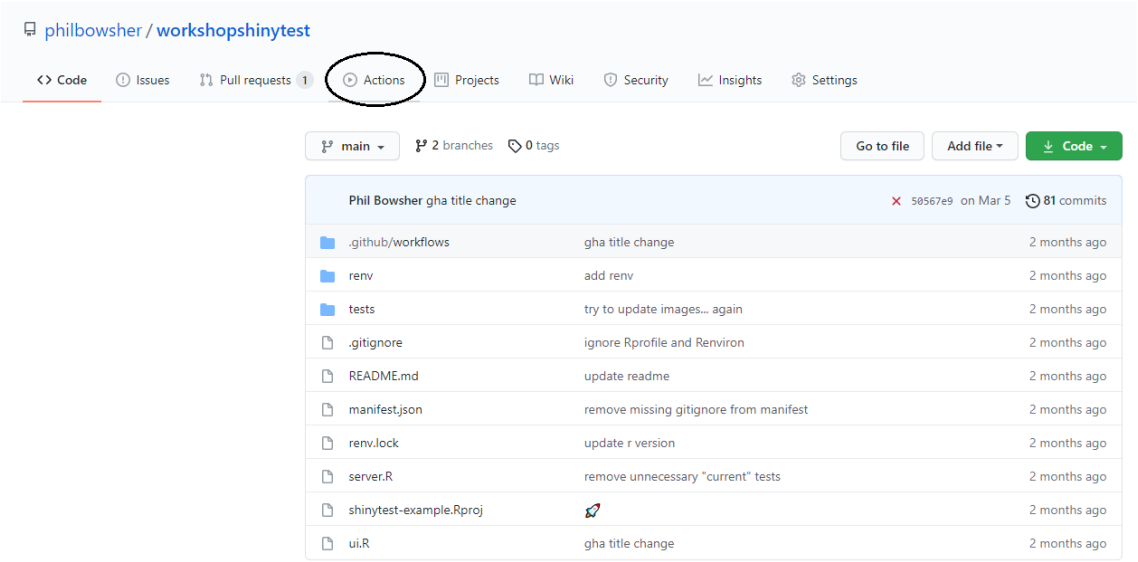
https://ymlthis.r-lib.org/

The fieldguide below provides a comprehensive source for finding options to set in YAML and learning what valid values are:

https://cran.r-project.org/web/packages/ymlthis/vignettes/yaml-fieldguide.html

Github Actions (CI/CD)

Github Actions (CI/CD) is a powerful tool for automating all your software workflows. This is a more advanced and more flexible workflow than the git-based deployment discussed above. Github Actions is currently free for open source repositories, cost-efficient for private repositories, and can be accessed via the repository that contains your Shiny app like this:



Below are ideas for things you may want to automate when deploying Shiny apps in increasing levels of complexity:

1. Deploy to RStudio Connect
2. Automate shinytest to run for your app and then, based on success, deploy an app to a Development/Staging server
3. Following approvals, deploy an app to production

Github Actions requires a manifest.json file as described in the previous section.

To set up Github Actions in a repository you will need to create a folder in the repository: .github/workflows/. This folder will contain 3 files described below:

- rstudio-connect-main.yaml - Deploys the main branch to RStudio Connect
- rstudio-connect-pr.yaml - Deploys the PR to RStudio Connect with a different name
- run-shinytest.yaml - Runs shinytest

You can see examples of the 3 files here:

https://github.com/colearendt/shinytest-playground/tree/main/.github/workflows

The first example shows how to deploy to RStudio Connect:

https://github.com/colearendt/shinytest-playground/blob/main/.github/workflows/rstudio-connect-main.yaml

Please note that your RStudio Connect server URL will need to be added to each file.

Now go to GitHub Actions on your repo and set the CONNECT_API_KEY secret. This is the API key generated in RStudio Connect by following this process:

https://docs.rstudio.com/connect/user/api-keys

Once you have the API key, go to the repository on GitHub with your shiny code. Navigate to Settings > Secrets. Create a "New Repository Secret" with the API key from RStudio Connect.

Make a git commit to the application and see the automatic deployment work!

CI testing with shinytest and GitHub Actions

Create a .github/workflows/run-shinytest.yaml file like the one here:

https://raw.githubusercontent.com/colearendt/shinytest-playground/main/.github/workflows/run-shinytest.yaml

If the tests succeed, the workflow run will pass. If the tests differ or fail, the workflow run will fail. It is worth noting and keeping in mind that tests / snapshots generated and run on different operating systems can cause failure due to image processing differences. This is highlighted in the issue below:

https://github.com/rstudio/shinytest/issues/382

Otherwise, if tests are failing, it means that something about the application has changed. Tests may need to be updated (or code fixed) to address the changes. To do this, you can run the tests locally or address any code changes necessary.

The example below shows how to deploy an application with a different name:

https://github.com/colearendt/shinytest-playground/blob/main/.github/workflows/rstudio-connect-pr.yaml

A similar process could be used to deploy the app to a different server before Production.

Contact & Summary

The information above highlights the exciting space of software development, and how well established tools in that ecosystems can help modernize clinical processes for reporting via Shiny applications.

Phil Bowsher

phil@rstudio.com

https://github.com/philbowsher