

Welcome to R in Production!

Please get **settled**,
grab a **coffee**,
connect to the **WiFi**
(**Posit Conf 2024 / conf2024**), and



complete the **prework** at
<https://github.com/posit-conf-2024/r-in-production>

R in production

The whole game

Hadley Wickham

Chief Scientist, Posit

August 2024



Hello
my name is

Hadley



Hello
my name is



Your turn

Introduce yourself to your neighbours.

What does “in production” mean to you?

Why did you choose to come to this workshop?



Show of hands

- Have you already run code in production?
- Have you used Posit Connect?
- Have you used shiny before?
- Have you used quarto before?
- Do use Git for your the majority of your data science projects?
- Have you used GitHub Actions?
- Have you used the usethis package?



What does in production mean?

- Code is run on another machine
(usually a linux server)
- Code is run repeatedly
(on a schedule, after another job, or on demand)
- The results are important and therefore are
a shared responsibility



09:00 - 10:30	“The whole game”
10:30 - 11:00	Coffee break
11:00 - 12:30	On another machine
12:30 - 13:30	Lunch break
13:30 - 15:00	Repeatedly
15:00 - 15:30	Coffee break
15:30 - 17:00	Shared responsibility



Two useful categories of production job

Batch	Interactive
R script, Rmd, qmd	shiny app, plumber API
Generate report, prep data, fit model, send notification	Explore data, score model
Run on schedule, or after another job completes	Run on demand
Can be computationally intensive	Best when computationally light
Usually single process	Might spawn multiple runners depending on demand + setup



Examples

- A dashboard might be a batch job (e.g. flexdashboard, quarto dashboards) or an interactive job (e.g. shinydashboard, shiny + bslib).
- If your interactive job is slow, a powerful and general technique is to pair it with a batch job that performs the heavy computation and saves the results.
- You could render RMarkdown reports interactively by creating a shiny app that calls `rmarkdown::render()` (this is effectively how parameterised reports work in Posit Connect).



Key verbs

- You **deploy** code from your development environment to a staging or production environment.
- A batch job is **executed** (if it's a script) or **rendered** (if it's a document) in the production environment. It's then **published** to somewhere you can see the results.
- An interactive job is just **deployed**, and is automatically **executed** when someone uses it.

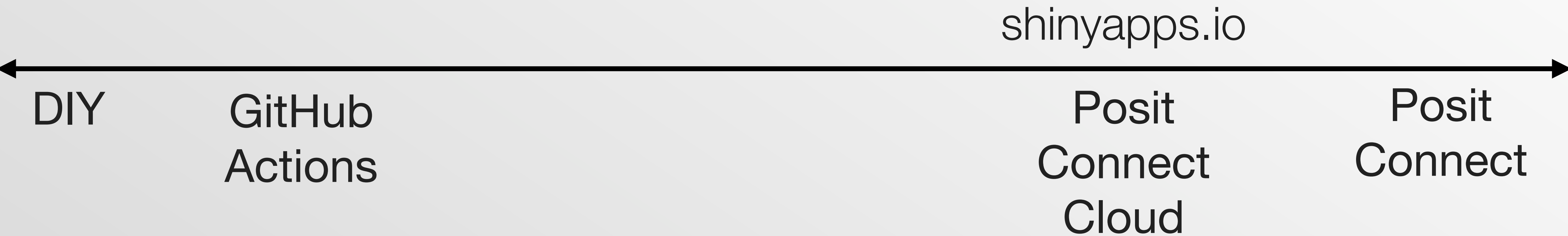


1. Practicalities
2. Living the project lifestyle
3. GitHub Actions
4. Posit Connect Cloud



Practicalities

The details of deploying code vary tremendously



rpubs.com
quartopub.com

Different tools support different job types

	Batch	Interactive
GHA	x	
Connect cloud	~	x
Posit Connect	x	x
shinyapps		x

I want to give you experience that you can apply elsewhere



rpubs.com
quartopub.com

GitHub actions

- Free for public repos, so great for personal projects and learning. Even if you don't use actions, you probably still use Git/GitHub.
- Tidyverse team maintains <https://github.com/r-lib/actions>, which we use primarily for package development, but also supports R in production on GitHub.
- Mid-level, so helps you understand what's going on, without getting too bogged down in the details.
- Only suitable for batch jobs

Posit Connect Cloud

- In alpha and therefore currently free. Will always have a free plan for public projects.
- Interface currently a hybrid of GHA and Posit Connect (but will grow towards Posit Connect).
- Most suited for interactive jobs.
- High-level: it takes care of all the details for you.

Commonalities

- In both, you have a Git repo that contains everything needed to run your job.
- This is a good workflow because it supports collaboration best practices.
- And I believe that git is a fundamental skill for all data scientists.
- Assumes you've adopted the project lifestyle

If you DIY:

- Use a docker container to capture specific OS version and system dependencies.
- Use rig to install R.
- Use pak to install R packages and system dependencies.
- Use PPPM to ensure that you get binaries.
(if you don't use rig.)

The project lifestyle

I assume you're already living the project lifestyle

- <https://www.tidyverse.org/blog/2017/12/workflow-vs-script/>
- A project = a directory = a git repo
- Git repo = source of truth
- All paths are relative to project top-level
- You frequently restart R & don't save/reload your workspace.



usethis provides tools to help you live this lifestyle

```
# Make usethis available in all future sessions
```

```
usethis::use_usethis()
```

```
# Tell RStudio not to save/reload sessions
```

```
usethis::use_blank_slate()
```

```
# (Positron does this by default)
```


And to tools to create new projects

```
# Create a new project
```

```
new_project("~/desktop/dashboard")
```

```
# Set up it to use git and GitHub
```

```
use_git()
```

```
use_github()
```

```
# Other useful functions you'll see today
```

```
use_description()
```

```
use_package()
```

```
use_github_pages()
```


Your turn

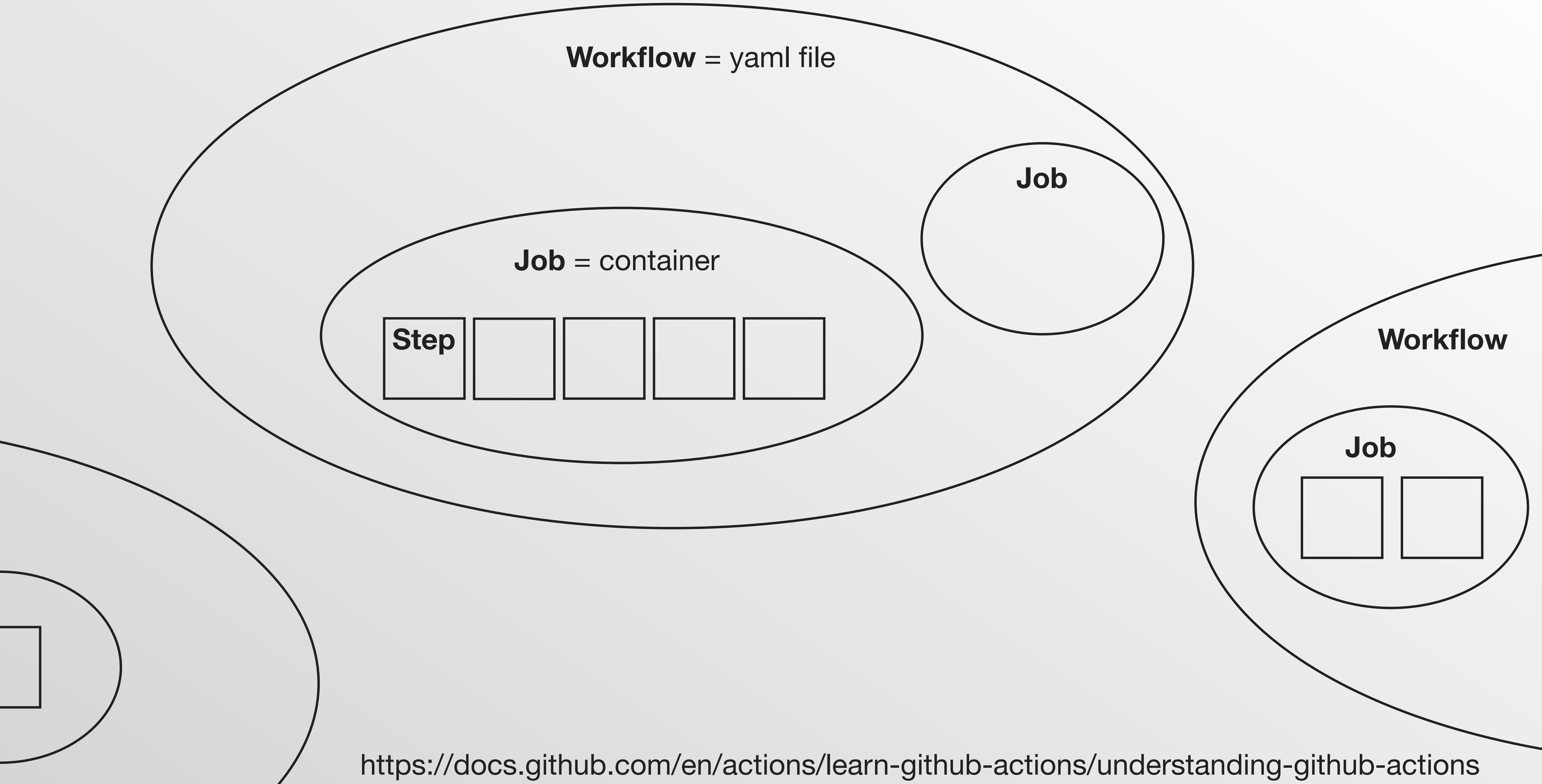
- Create a new project called **diamonds**.
- Publish it to GitHub.
- (We'll add some content shortly)

GitHub actions

Personal examples

- <https://github.com/hadley/available-work>: scrapes an artist's website and notifies me when new work is available.
- <https://github.com/hadley/houston-pollen>: scrapes daily pollen data and aggregates it into a yearly parquet file.
- <https://github.com/hadley/cran-deadlines>: turns CRAN deadline data into a Quarto dashboard.

GitHub action basics



We recommend the following naming convention

- If the filename is **foo.yaml**
- Then the action should be named **foo.yaml**
- And if you only have one job, it should be named **foo**

There are three fields you'll see in every workflow file

In `.github/workflows/render.yml`

name: `render.yml`

when to run

on:

what to do

jobs:

on tells GHA when to run your code

on:

```
# Run when code pushed to GitHub
```

```
push:
```

```
# Run when the user asks for it
```

```
workflow_dispatch:
```

```
# Run at 9:23pm Monday-Friday
```

```
schedule:
```

```
- cron: '23 21 * * 1-5'
```

Advice

- Usually want `push`, `workflow_dispatch`, and `schedule`. Might also want `pull_request`.
- Use <https://crontab.guru/> or an LLM to design/check your cron tab specification.
- Always include a comment describing it in human language!
- Scheduled job will only run for 60 days after last commit to repo.
- Always use a minute offset
`sample(setdiff(0:59, seq(0, 60, by = 5)), 1)`

Generate crontab specifications for:

- 9am every Friday
- On the first and 15th of every month
- Every 30 minutes on the weekend
- Every 3 days at 10:23am
- Every hour during the work week (i.e. 9am-5pm Mon-Fri)

Hint: LLM's do great at these

9am in WHAT TIMEZONE?


```
library(lubridate)
```


```
my_time ← ymd_hm("2024-08-15 09:00", tz = "America/Chicago")
```


```
with_tz(my_time, "UTC")
```


```
# As far as I can tell you can't account for daylight savings time
```


Posit Connect is a clear winner here



Info


Access


Runtime


Schedule


Tags


Vars

☒ Schedule output for default

Timezone

(GMT-07:00) America - Los Angeles

Start date & time

Jul 28, 2023

1

:

25

am

pm

[Reset to Local Time](#)

jobs field tells GHA what to do

jobs:

scrape:

runs-on: ubuntu-latest

permissions:

contents: write

steps:

- uses: actions/checkout@v4

- uses: r-lib/actions/setup-r@v2

with:

use-public-rspm: true

- uses: r-lib/actions/setup-r-dependencies@v2

- name: Fetch latest data

run: Rscript scrape.R

- name: Collapse into yearly parquet files

run: Rscript collapse.R

- uses: stefanzweifel/git-auto-commit-action@v5

What container should the job use

There are usually three phases

```
- uses: actions/checkout@v4
- uses: r-lib/actions/setup-r@v2
  with:
    use-public-rspm: true
- uses: r-lib/actions/setup-r-dependencies@v2
```

Setup

```
- name: Fetch latest data
  run: Rscript scrape.R

- name: Collapse into yearly parquet files
  run: Rscript collapse.R
```

Execute

```
- uses: stefanzweifel/git-auto-commit-action@v5
```

Public

Common setup steps

```
# check out your repo
```

```
- uses: actions/checkout@v4
```

```
# install R
```

```
- uses: r-lib/actions/setup-r@v2
```

Powered by rig

```
  with:
```

```
    use-public-rspm: true
```

```
# install dependency from description
```

```
- uses: r-lib/actions/setup-r-dependencies@v2
```

Powered by pak

```
# install dependency from renv lockfile
```

```
- uses: r-lib/actions/setup-renv@v2
```

Powered by renv

```
# install pandoc
```

```
- uses: r-lib/actions/setup-pandoc@v2
```

```
# install quarto
```

```
- uses: quarto-dev/quarto-actions/install-quarto@v1
```

Common execution steps

- name: Fetch latest data
run: Rscript scrape.R
- name: Render Rmarkdown
shell: Rscript {0}
run: rmarkdown::render("myfile.Rmd")
- name: Render Quarto directory
run: quarto render

Common publishing steps

- `uses: stefanzweifel/git-auto-commit-action@v5`
- `name: Publish to GitHub pages 🚀`
`if: github.event_name ≠ 'pull_request'`
`uses: JamesIves/github-pages-deploy-action@v4.5.0`
`with:`
 - `branch: gh-pages`
 - `folder: docs`

General advice

- No one remembers what the steps look like. You either copy them from an existing repo or hope that an LLM gives you good advice.
- We have a bunch of examples at <https://github.com/r-lib/actions/tree/v2-branch/examples>
- Comment heavily so when you come back it, you can remember what you were trying to do.
- Don't expect to get it right on the first try!

Your turn

- Find the actions in each of the repos below. What do they do? What's the same and what's different? Can you figure out how they determine which R packages are needed?
- <https://github.com/hadley/available-work>
- <https://github.com/hadley/houston-pollen>
- <https://github.com/hadley/cran-deadlines>

These jobs describe their dependencies with DESCRIPTION

```
use_description()
```

```
use_package("ggplot2")
```

```
use_package("dplyr")
```

Fits in naturally with usethis project workflow

```
create_project("~/desktop/diamonds")
use_git()
use_github()
use_description()
use_package("ggplot2")
use_github_action(url = "https://github.com/posit-conf-2024/r-in-
production/blob/main/render-rmd.yaml")
# Turn on GitHub pages
usethis::use_github_pages()
```

Your turn

- Open your diamonds project.
- Add an Rmd that draws a plot of the ggplot2 diamonds dataset. Include the current date and time in the output.
- Check that it works locally then push your code to GitHub.
- Add `.github/workflows/render.yml` and create a DESCRIPTION that defines your dependencies.
- Push to GitHub then iterate until it works :)
- Stretch goals on the next slide

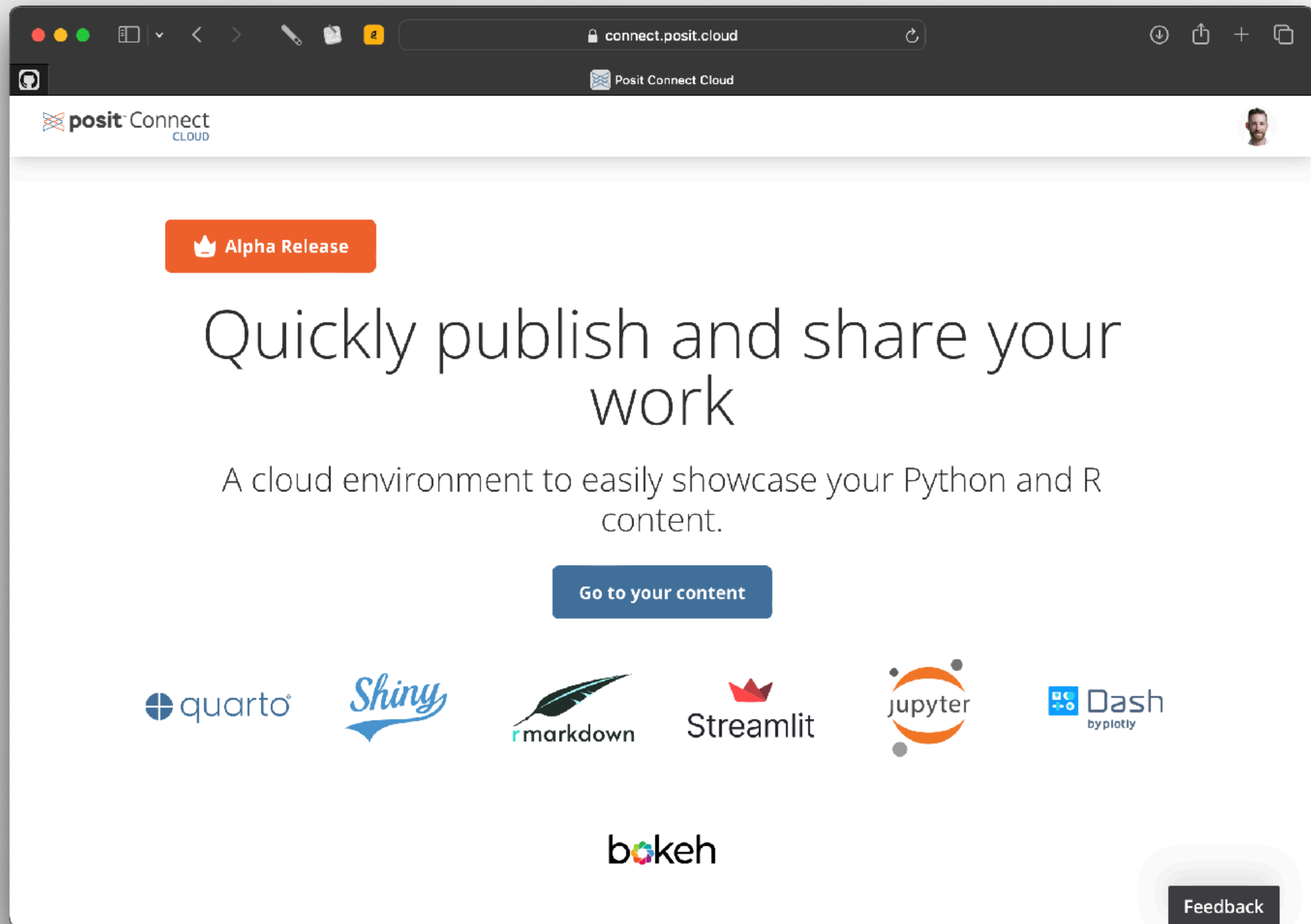
Stretch goals

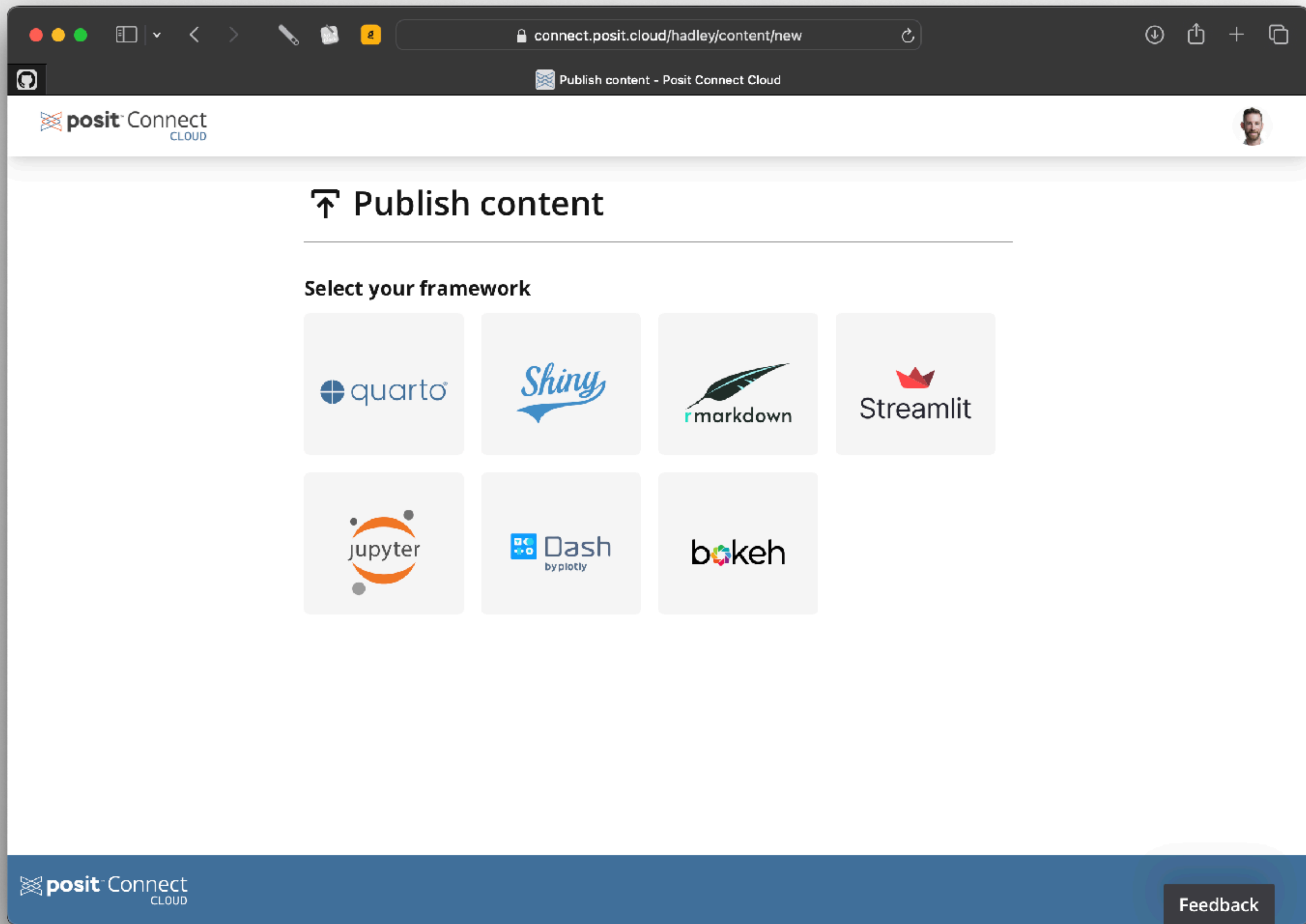
- Replace **on: push** with **on: workflow_dispatch**. Push to GitHub, and confirm that the action now doesn't run automatically. Manually trigger the workflow from the actions page.
- Create another Rmd and render it too. Add a link to it from index.Rmd.
- Convert your .Rmd to a .qmd and render it with quarto. What do you need to change?

Posit connect cloud

Compared to GitHub Actions

- Takes care of many more details: it picks a docker image for you; you select from a small set of possible job types.
- Either just works or won't work for your use case.
- Current key use case is shiny apps. Does support quarto batch jobs, but not yet compelling because there's no scheduling.
- Uses different/better dependency installation system (manifest.json will come to pak/GHA in the future)
- Fast!





Your turn

- Create a new madlibs project.
- `use_git()`; `use_github()`.
- Copy in madlibs.R. Commit.
- `rsconnect::writeManifest()`. Commit.
- Push to GitHub.
- Deploy with connect cloud & view it.
- Stretch goals on next slide.

Stretch goals

- Improve the madlib story or instructions.
- Remove the button and have the output update live.
- Use `shinyvalidate` to check the inputs.
- (Don't know how? Try asking an LLM.)

Differences between Connect and Connect Cloud

- Scheduled & parameterised reports.
- Can also host APIs and RMarkdown docs
- Automatically send emails (hard to do these days).
- Highest-level interface: automatically detects job type from file structure and calls `writeManifest()` when you click deploy.
- Costs \$\$\$ & requires IT integration, but seamlessly integrated with your auth so stuff just works.

Want to learn more? Head to the lounge