Welcome to **R in Production**!

Please get **settled**,
grab a **coffee**,
connect to the **WiFi**, and
complete the **prework** at
https://github.com/posit-conf-2025/r-production

R in production

The whole game

Hadley Wickham

Chief Scientist, Posit

September 2025

Hello

my name is

Hadley

Hello

my name is

Davis

Charlie

Tom

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)
- Code (and data) is a shared responsibility
  (because it’s important and someone cares if it breaks)

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

1. Production projects
2. Deployment options
3. GitHub Actions
4. Posit Connect Cloud

Production projects

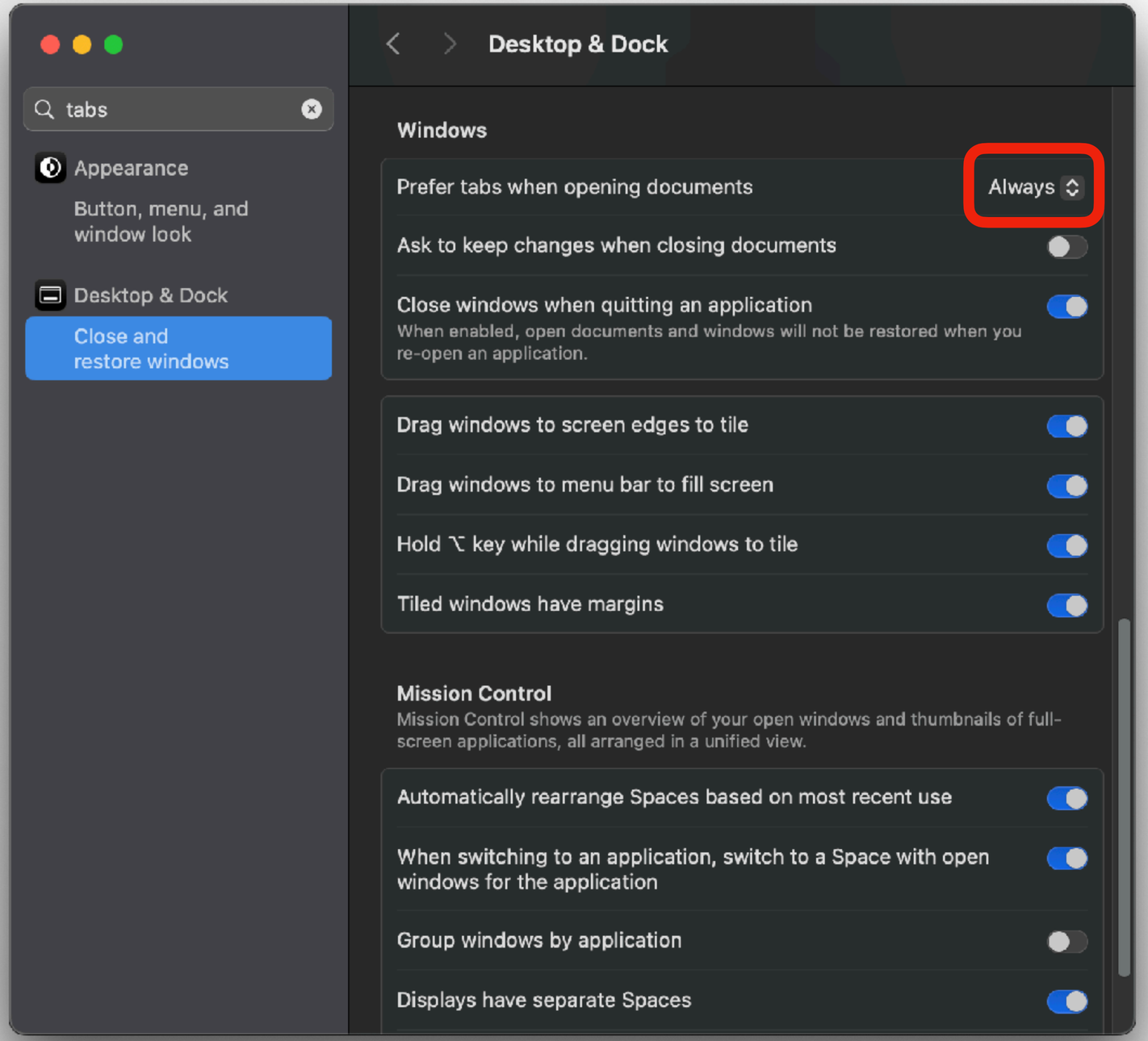
https://r-in-production.org/project-life.html

1. A project must be a self-contained directory
2. There must be a single source of truth (and that must be **git**)
3. Deployment must be automated (NO CLICKING BUTTONS!)

Self-contained directory

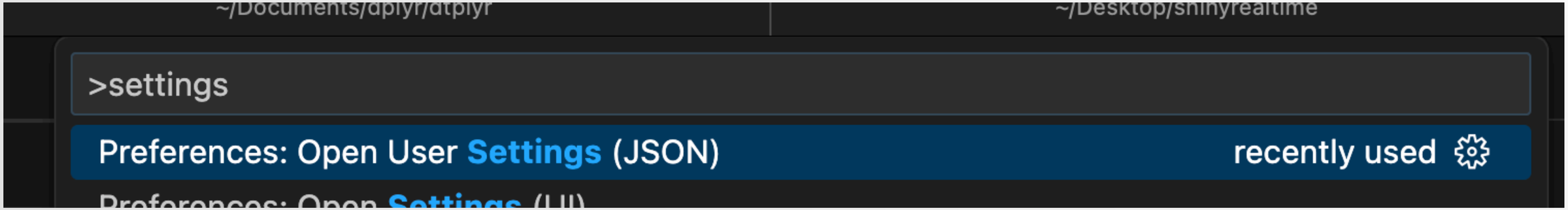
What is a project?

- Used to be very clear with RStudio: a directory with a .Rproj file
- A little less clear with Positron; now any directory that you happen to have opened.
- But the idea remains the same: put all the stuff you need for a project in one place.
- (Project navigation is also a bit different since Positron tends to be full screen and you only have one open)



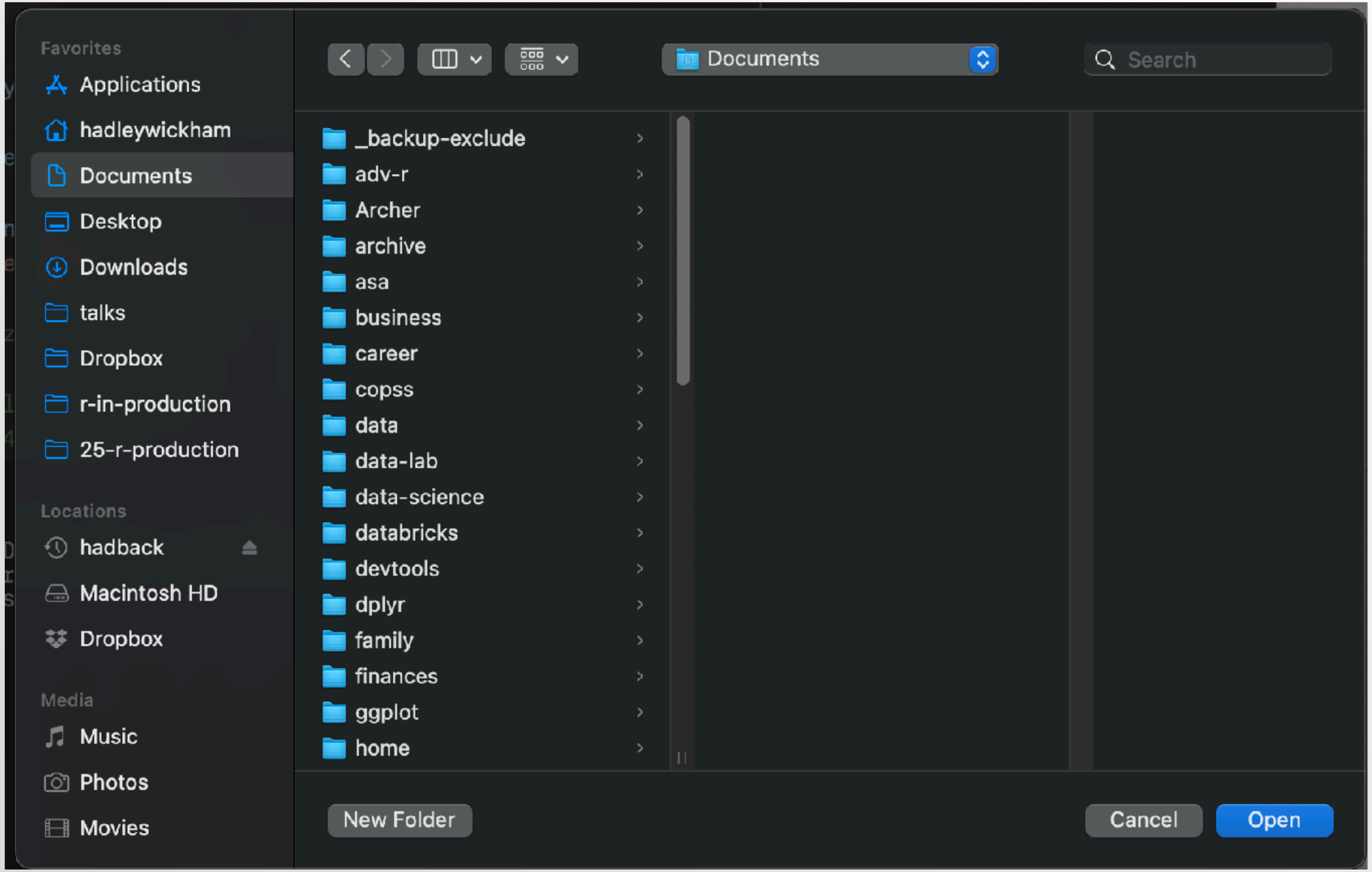
Hopefully not needed with future Positron release

Cmd + Shift + P

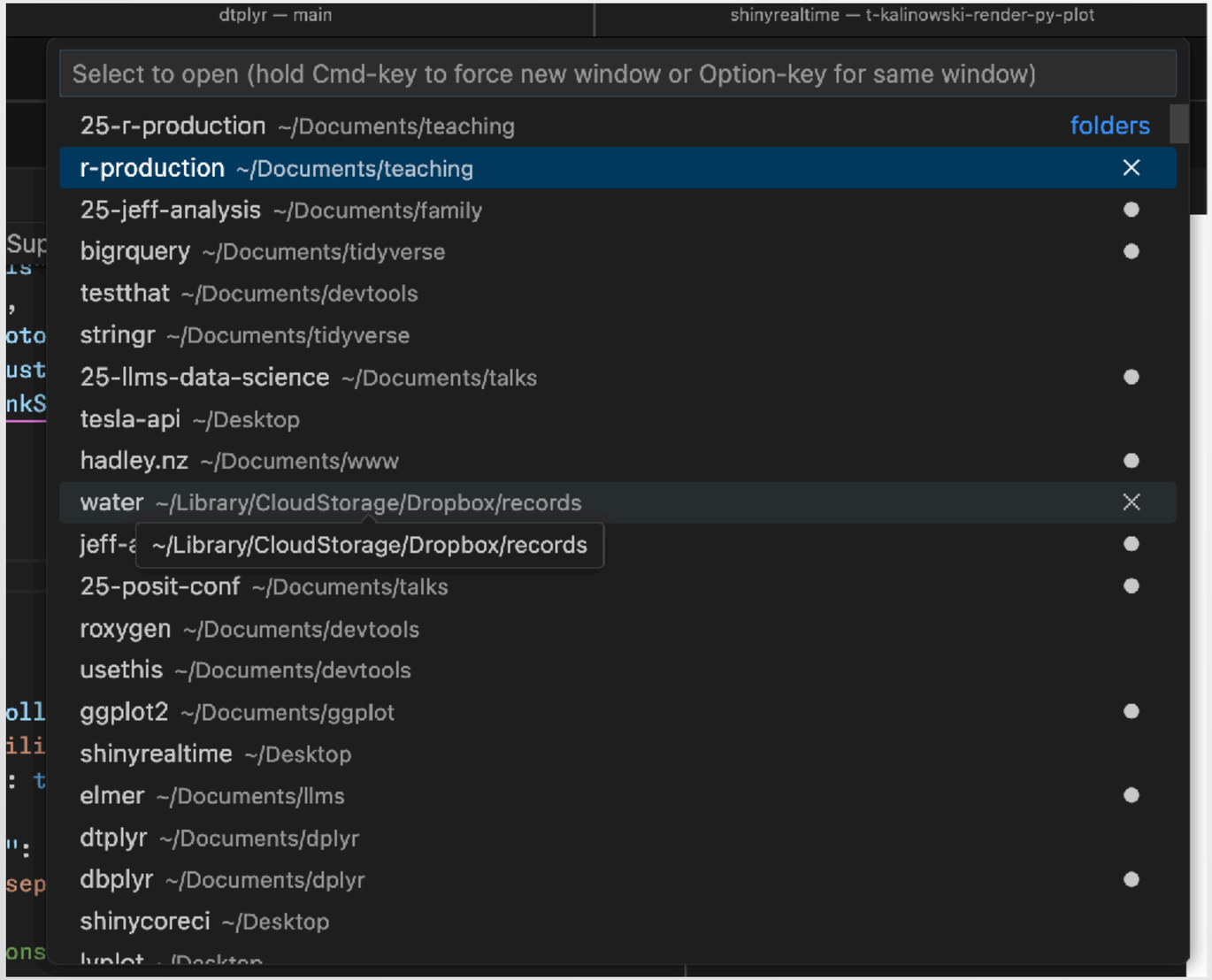



```
"window.nativeTabs": true,
"window.openFoldersInNewWindow": "on",
"window.title": "${rootName}${separator}${activeRepositoryBranchName}"
```

Cmd + O


Favorites
Applications
hadleywickham
Documents
Desktop
Downloads
talks
Dropbox
r-in-production
25-r-production
Locations
hadback
Macintosh HD
Dropbox
Media
Music
Photos
Movies
Documents
Search
_backup-exclude
adv-r
Archer
archive
asa
business
career
copss
data
data-lab
data-science
databricks
devtools
dplyr
family
finances
ggplot
home
New Folder
Cancel
Open

Ctrl + R


dtplyr — main
shinyrealtime — t-kalinowski-render-py-plot
Select to open (hold Cmd-key to force new window or Option-key for same window)
25-r-production ~/Documents/teaching folders
r-production ~/Documents/teaching
25-jeff-analysis ~/Documents/family
bigrquery ~/Documents/tidyverse
testthat ~/Documents/devtools
stringr ~/Documents/tidyverse
25-llms-data-science ~/Documents/talks
tesla-api ~/Desktop
hadley.nz ~/Documents/www
water ~/Library/CloudStorage/Dropbox/records
~/Library/CloudStorage/Dropbox/records
25-posit-conf ~/Documents/talks
roxygen ~/Documents/devtools
usethis ~/Documents/devtools
ggplot2 ~/Documents/ggplot
shinyrealtime ~/Desktop
elmer ~/Documents/llms
dtplyr ~/Documents/dplyr
dbplyr ~/Documents/dplyr
shinycoreci ~/Desktop

Lots of ways to create projects

- Create an empty folder
- Use Positron helper
- usethis::create_project()

Your turn

Apart from your R codes, what else needs to go into a project to ensure that it’s self contained?

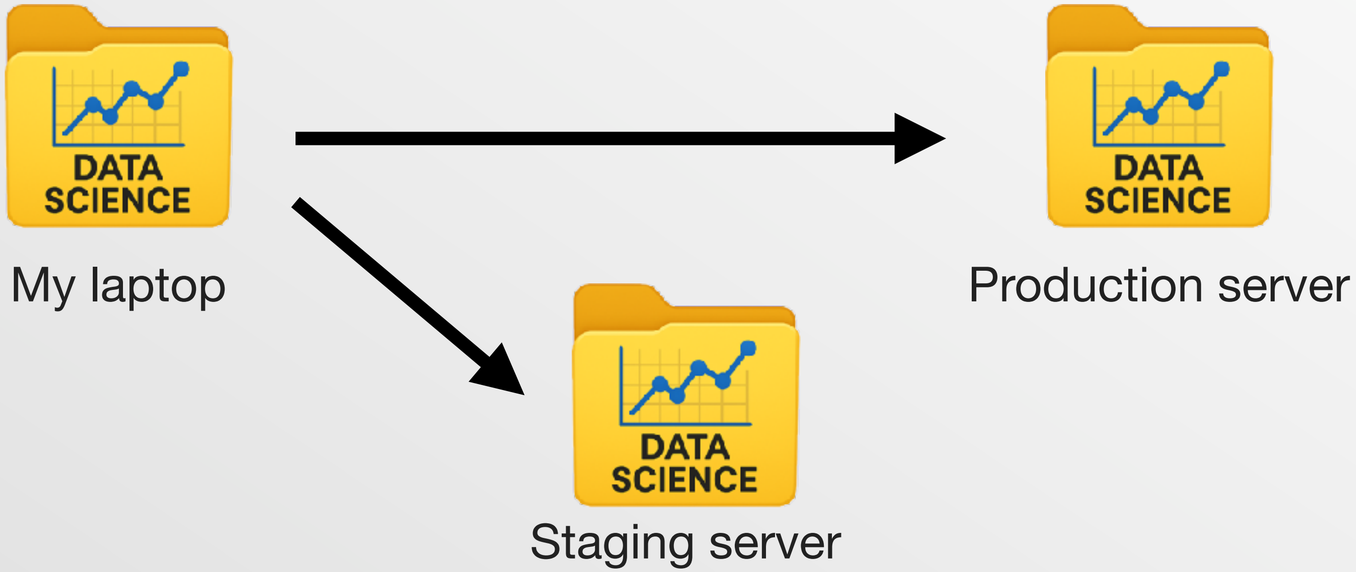
Single source of truth

Life starts simple

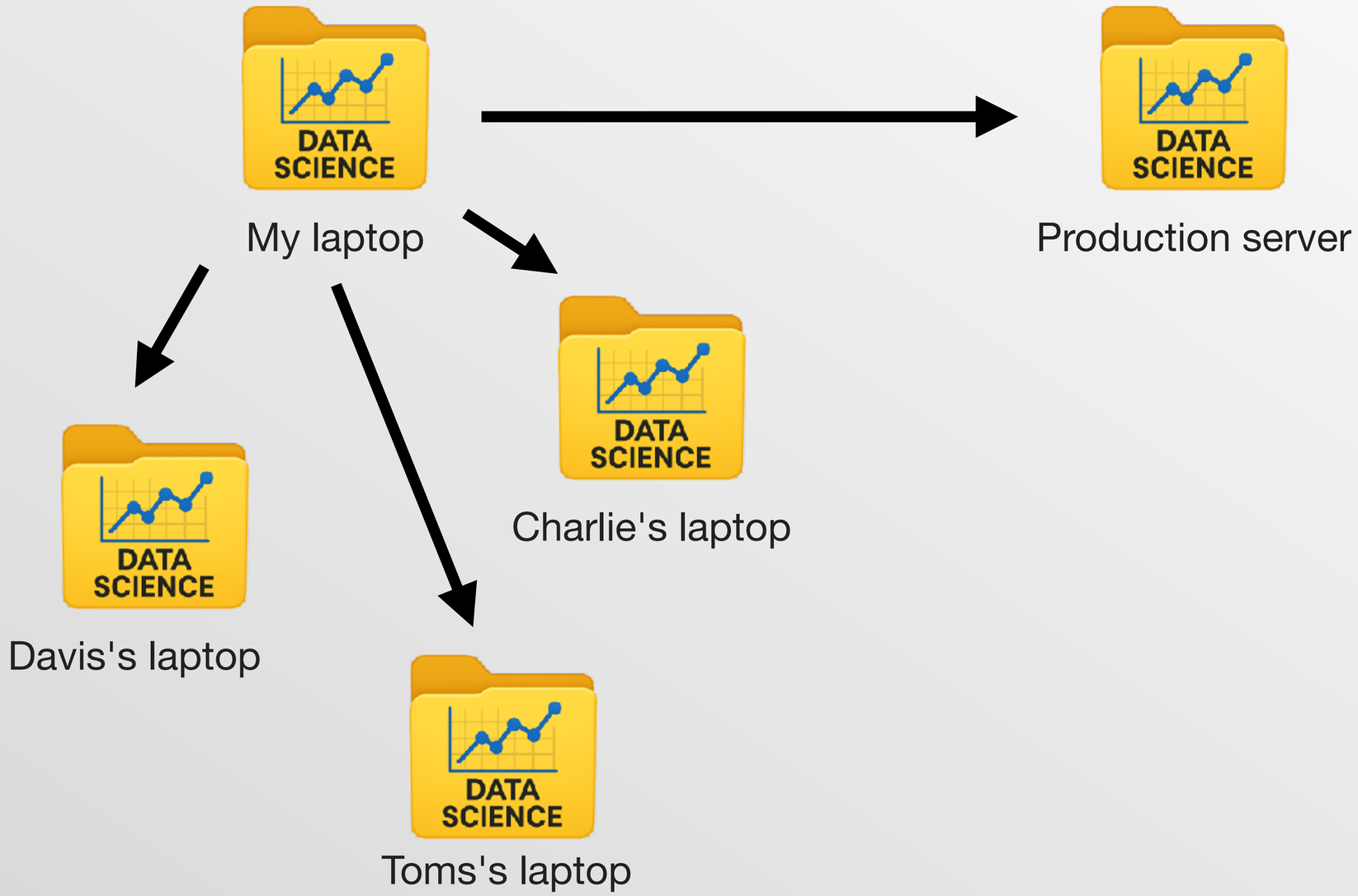
My laptop

Production server

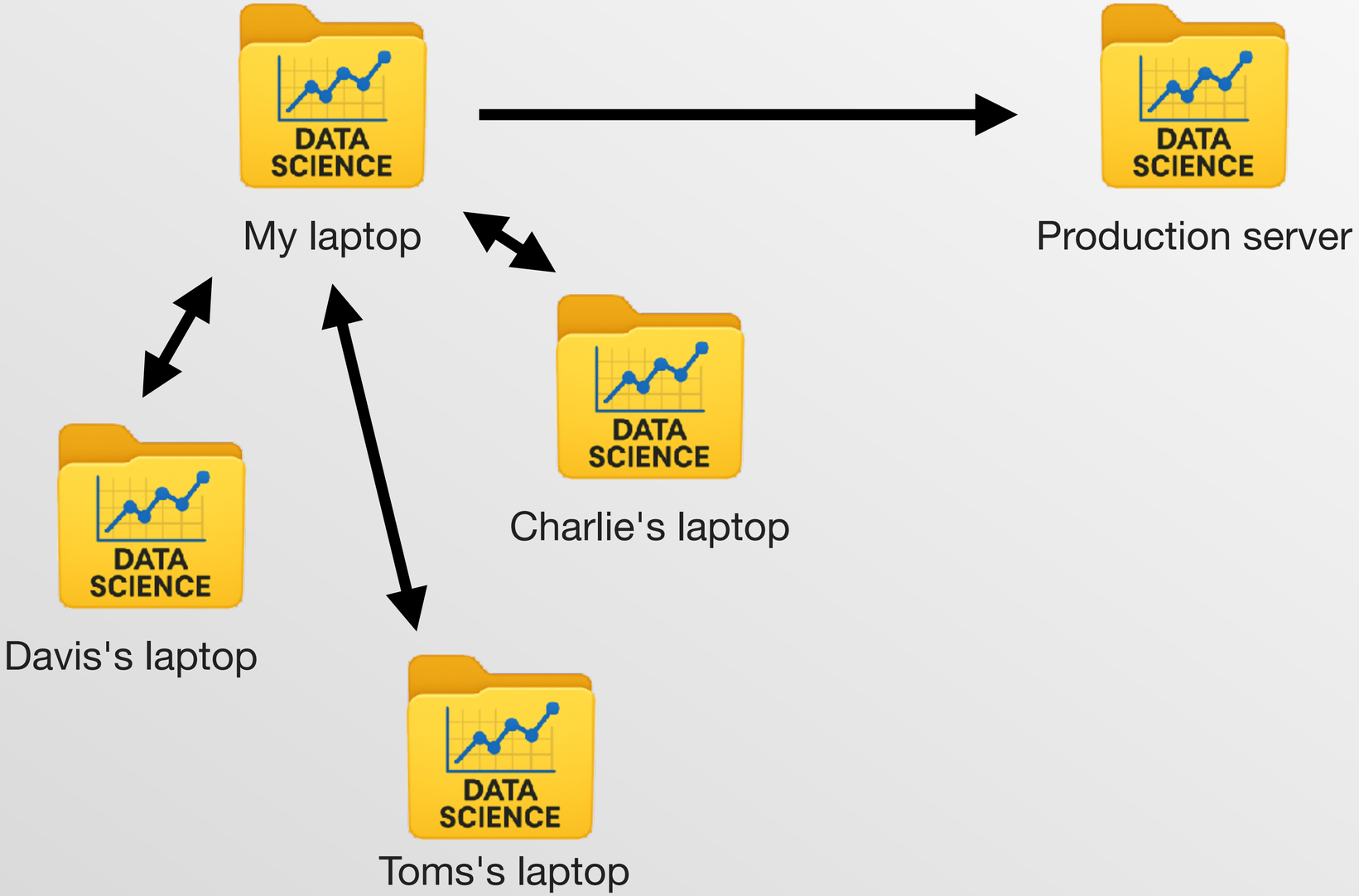
You'll have 3 copies if you also have a staging server



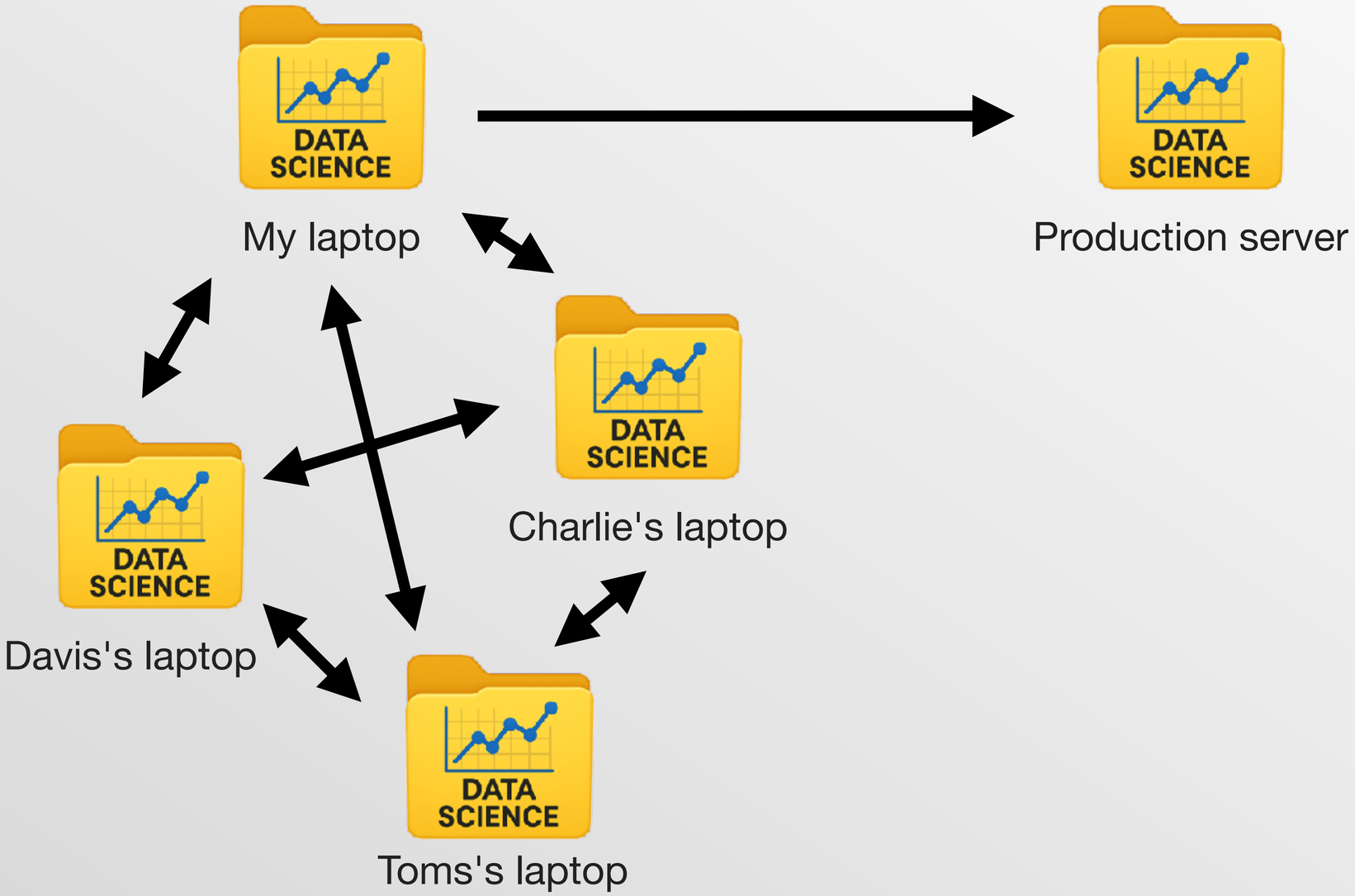
But what if it’s a team project?



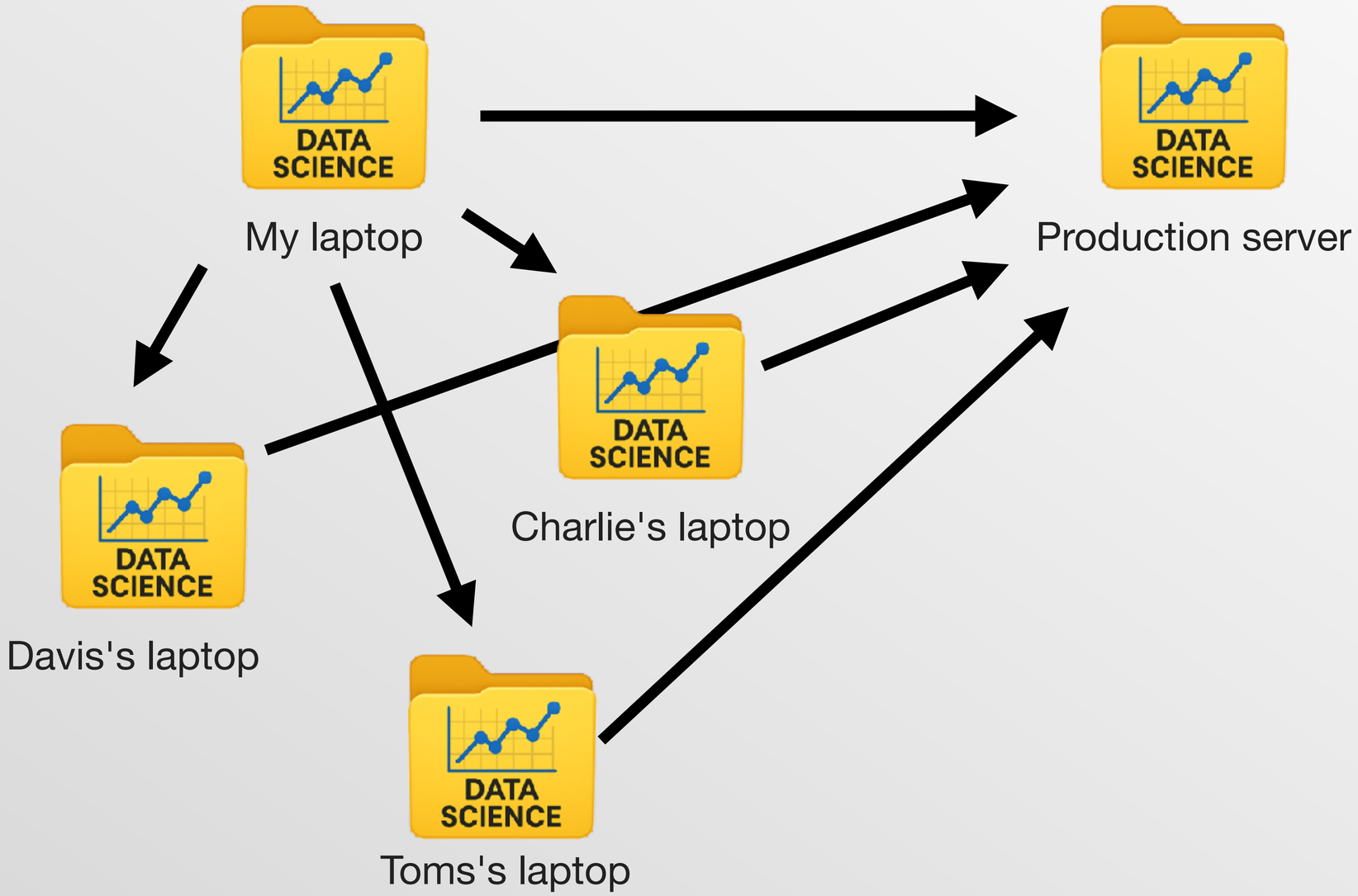
Does everyone have to get the results back to me?



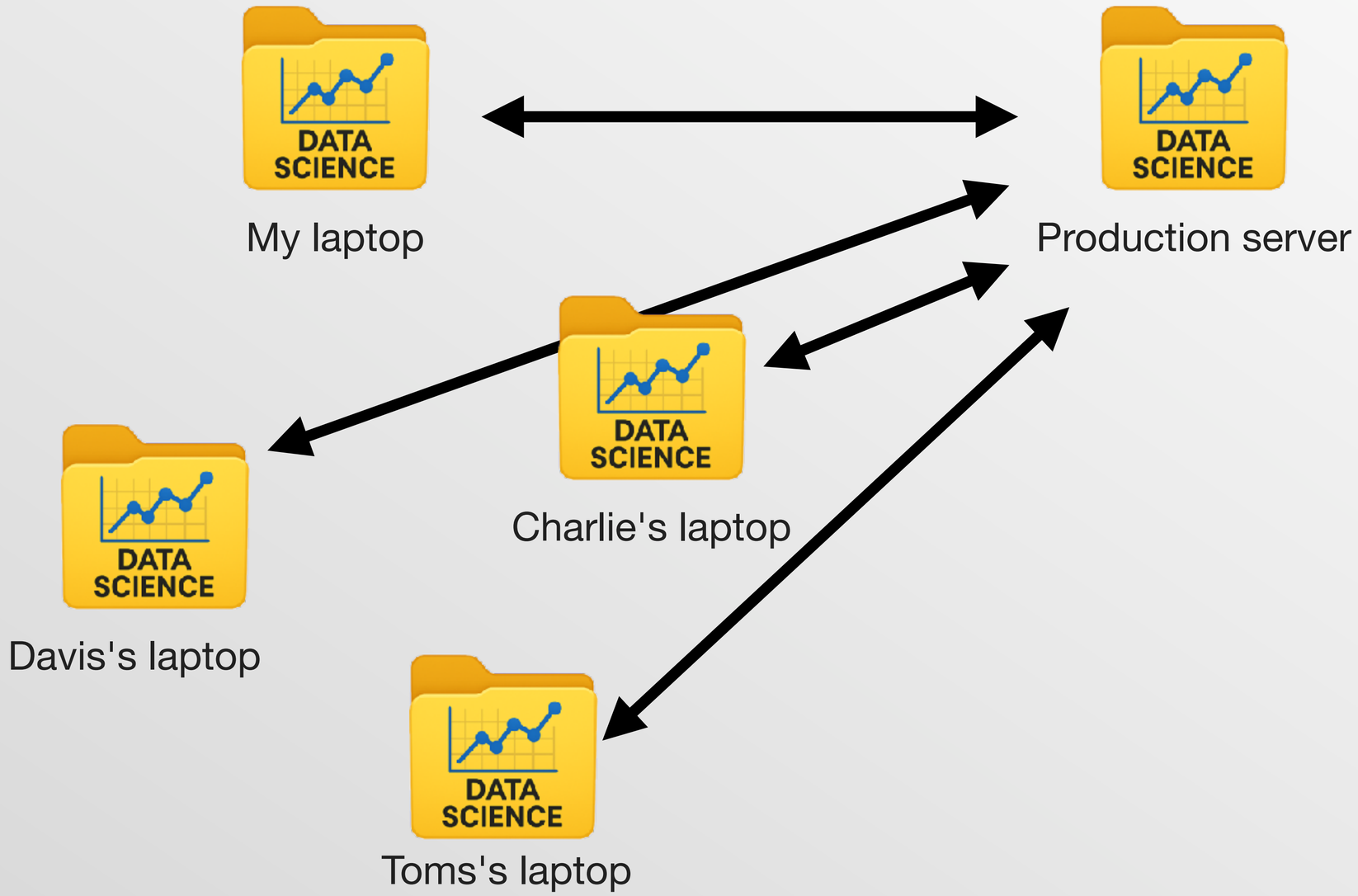
Can they also share with each other?



Maybe they can publish to production too?



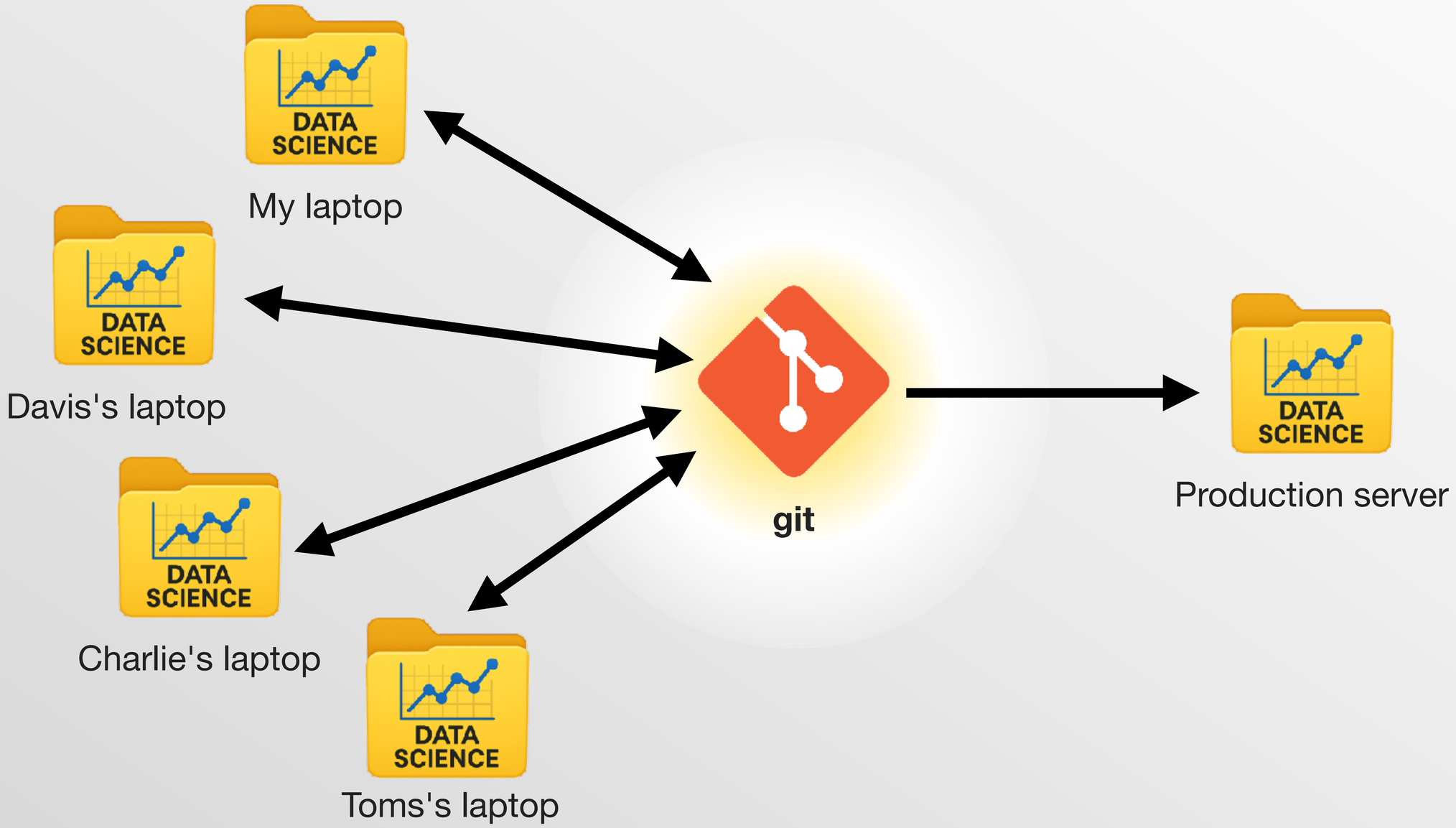
Maybe we use production as the source of truth?



This way lies madness

There is only one true way

A central git repo must be the source of truth



This is why data scientists **must** know git

You also need something like GitHub

- Track issues
- Create an review pull requests
- Readable project documentation with a README
- Automated testing (e.g. CI/CD)

- (Something like GitHub = GitLab, Bitbucket, Azure DevOps.
  If you don’t have it, advocate for it!)

Project first steps

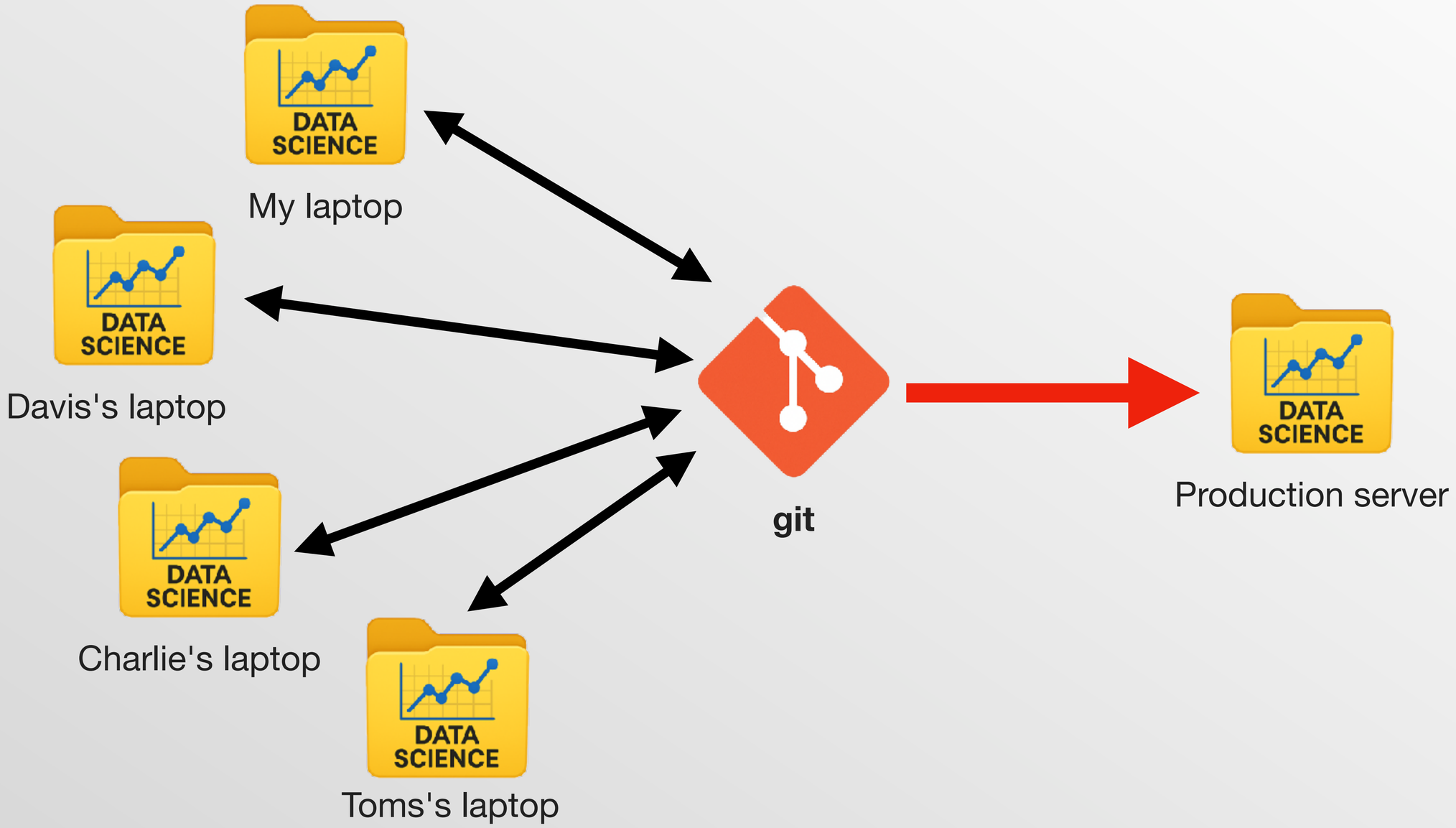
```
# Create a new project
create_project("~/desktop/dashboard")

# Set up it to use git and GitHub
use_git()
use_github()
```

Your turn

- Create a new project called **diamonds**
- Set up git
- Publish it to GitHub
- (We’ll add some content shortly)

Automated deployment

This our next topic



Deployment options

The details of deploying code vary tremendously

DIY

GitHub Actions

Posit Connect Cloud

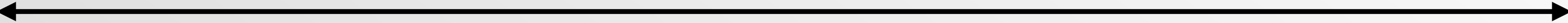
Posit Connect

shinyapps.io

rpubs.com

quartopub.com

I want to give you experience that you can apply elsewhere



DIY

GitHub Actions

Posit Connect Cloud

Posit Connect

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).

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

- Eternal free plan for public projects.
  Various paid options for personal + org usage.
- Growing towards support for all Posit Connect features.
- Most useful for interactive jobs
  (scheduled + parameterised reports coming soon)
- High-level, it takes care of all the details for you.

If you DIY:

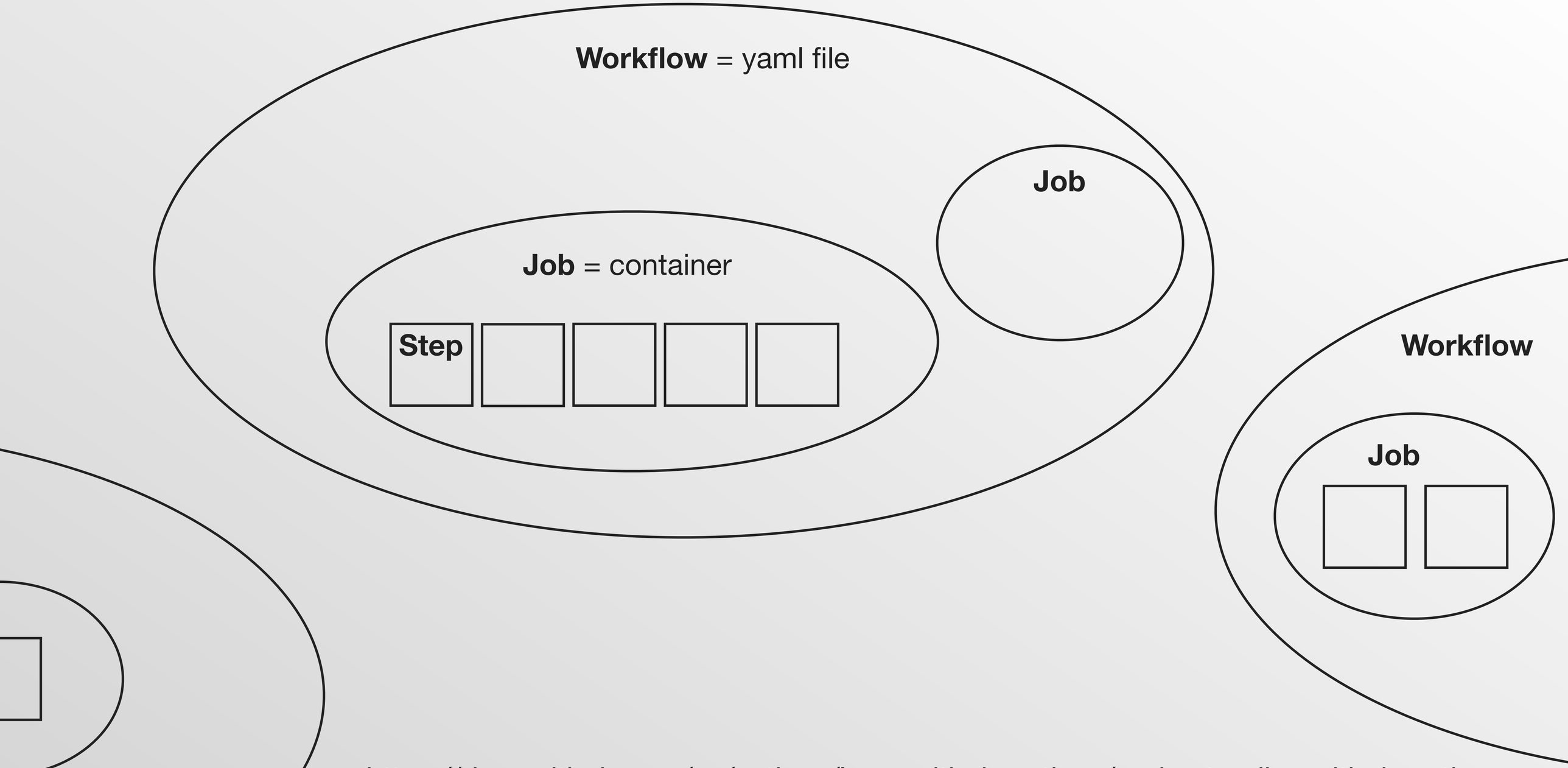
- Use a docker container to capture specific OS version and system dependencies.
- Use rig to install R.
- Use pak to install R packages and system dependencies.
- Use P3M to ensure that you get binaries.
  (e.g. pak::repo_add(CRAN = "PPM@latest"); rig does this for you)

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



https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

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

```yaml
# In .github/workflows/render.yml

name: render.yml

# when to run
on:

# what to do
jobs:
```

filename = **foo.yaml**
action name = **foo.yaml**
job name = **foo**

on tells GHA when to run your code

```yaml
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:

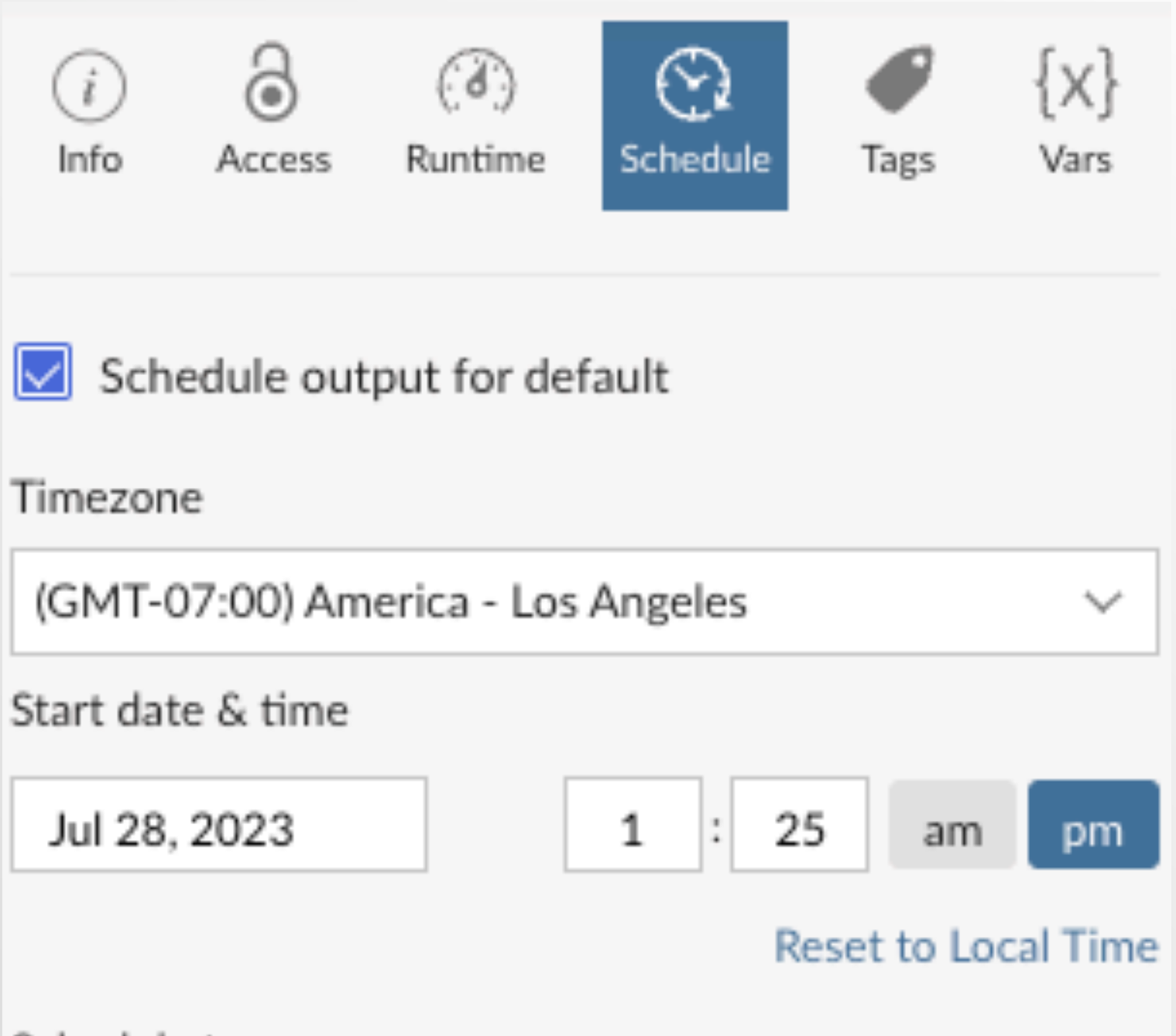
- 9am every Friday
- On the first and 15th of every month
- Every 30 minutes on the weekend
- Every 3 days at 10:23am
- Every hour during the work week (i.e. 9am-5pm Mon-Fri)

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



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
```

Publish

Common setup steps

```yaml
# check out your repo
- uses: actions/checkout@v4
# install R
- uses: r-lib/actions/setup-r@v2
  with:
    use-public-rspm: true
# install dependency from description
- uses: r-lib/actions/setup-r-dependencies@v2
# install dependency from renv lockfile
- uses: r-lib/actions/setup-renv@v2
# install pandoc
- uses: r-lib/actions/setup-pandoc@v2
# install quarto
- uses: quarto-dev/quarto-actions/install-quarto@v1
```

Powered by rig

Powered by pak

Powered by renv

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/houston-pollen

- https://github.com/hadley/available-work

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_package("rmarkdown")
use_github_action(url = "https://github.com/posit-conf-2025/r-
production/blob/main/render-rmd.yaml")
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. (What file name should your Rmd use?)
- 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

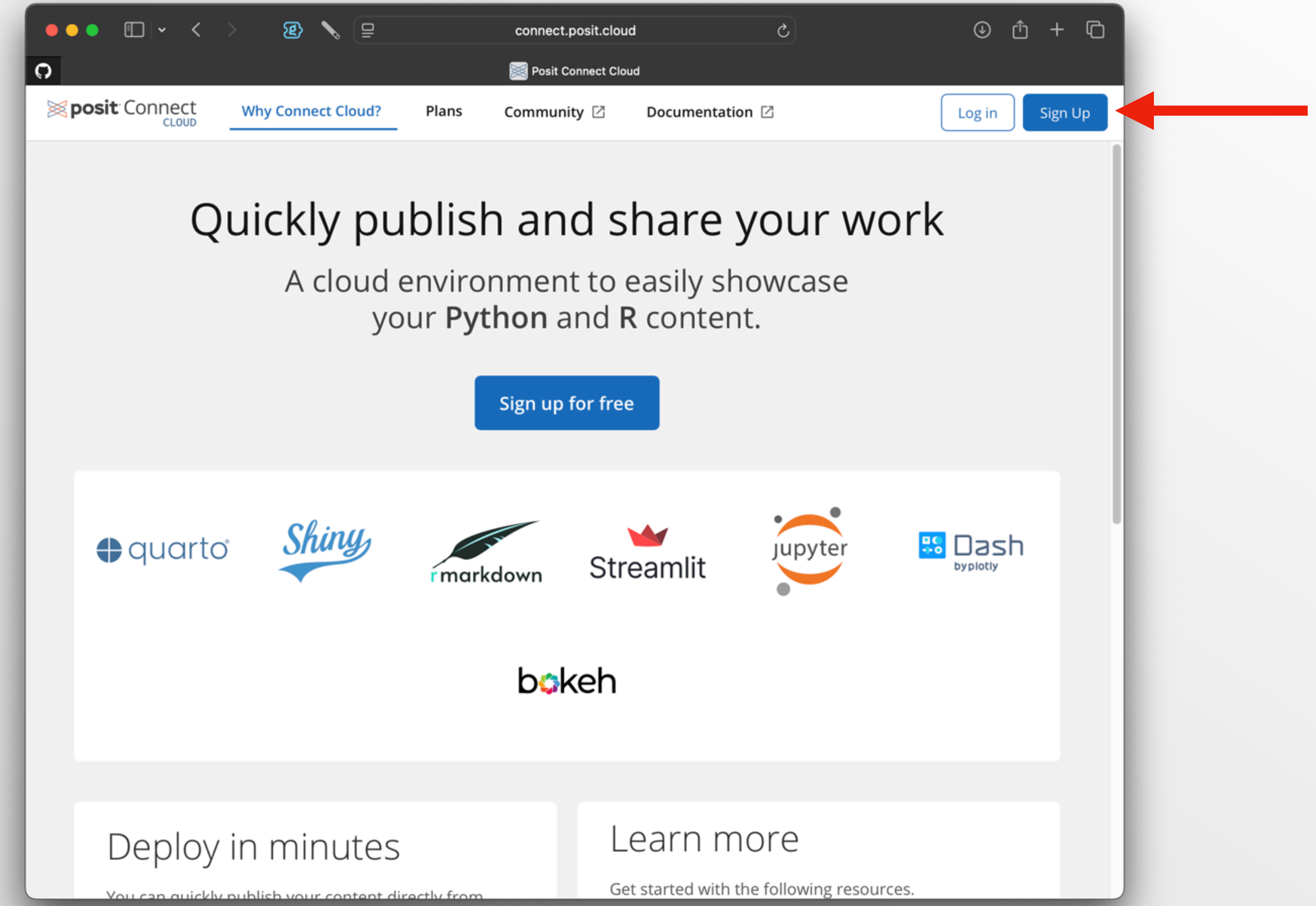
https://connect.posit.cloud/

Compared to GitHub Actions

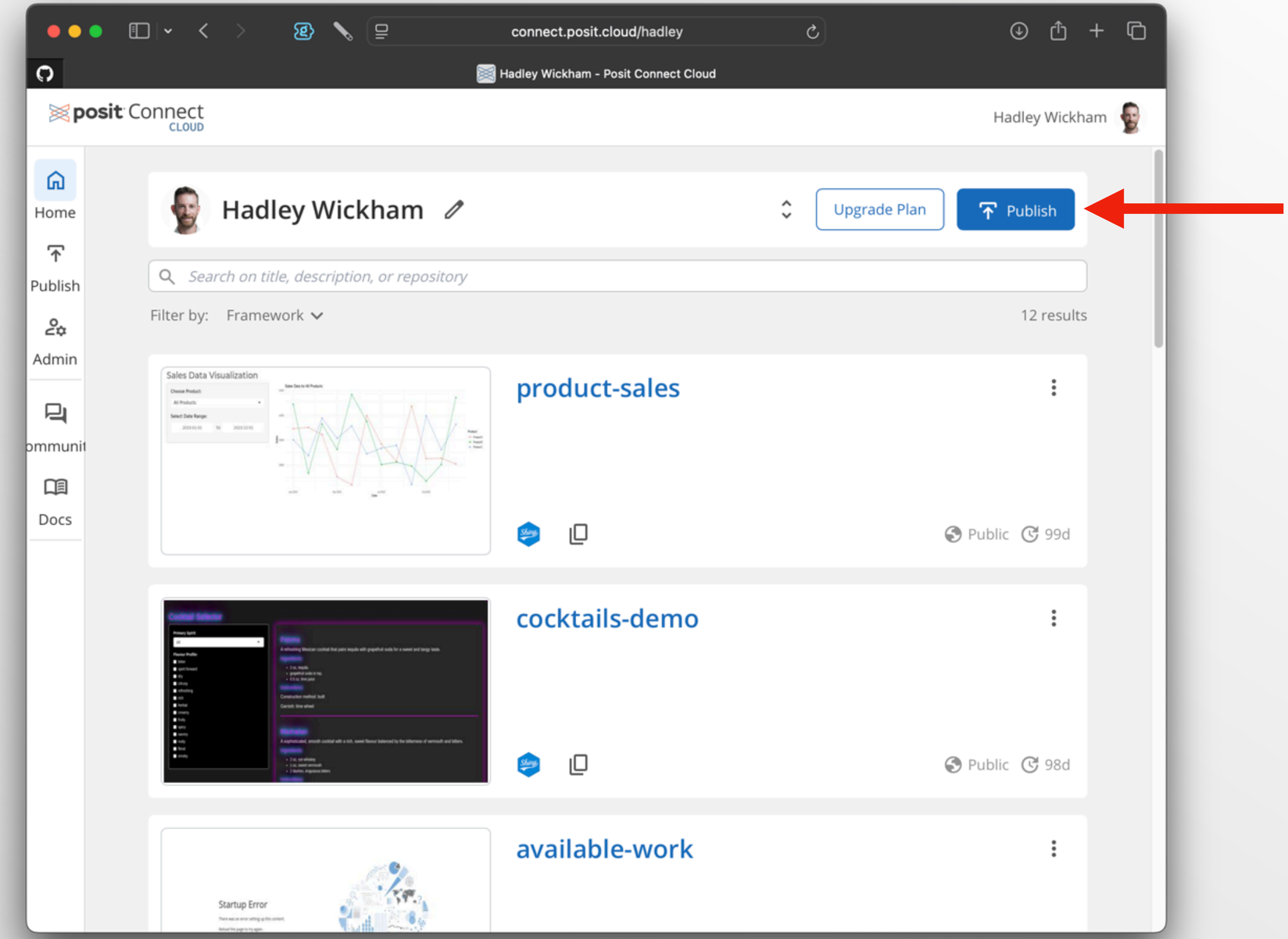
- Takes care of many more details: it picks a docker image for you; you select from a small set of possible job types.
- Either just works or won’t work for your use case.
- Current key use case is shiny apps. Does support quarto batch jobs, but not yet compelling because there’s no scheduling.
- Uses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

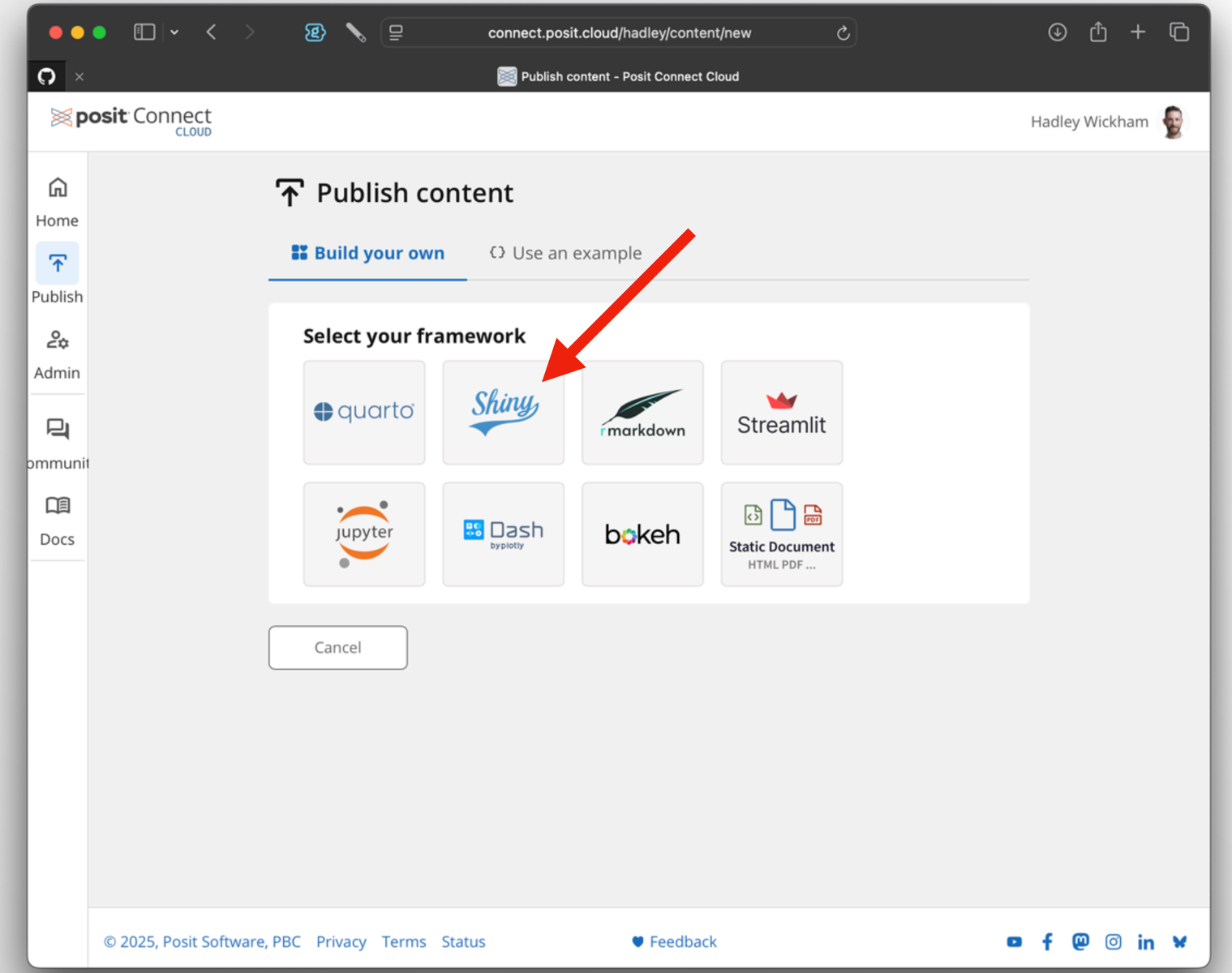
Personal examples

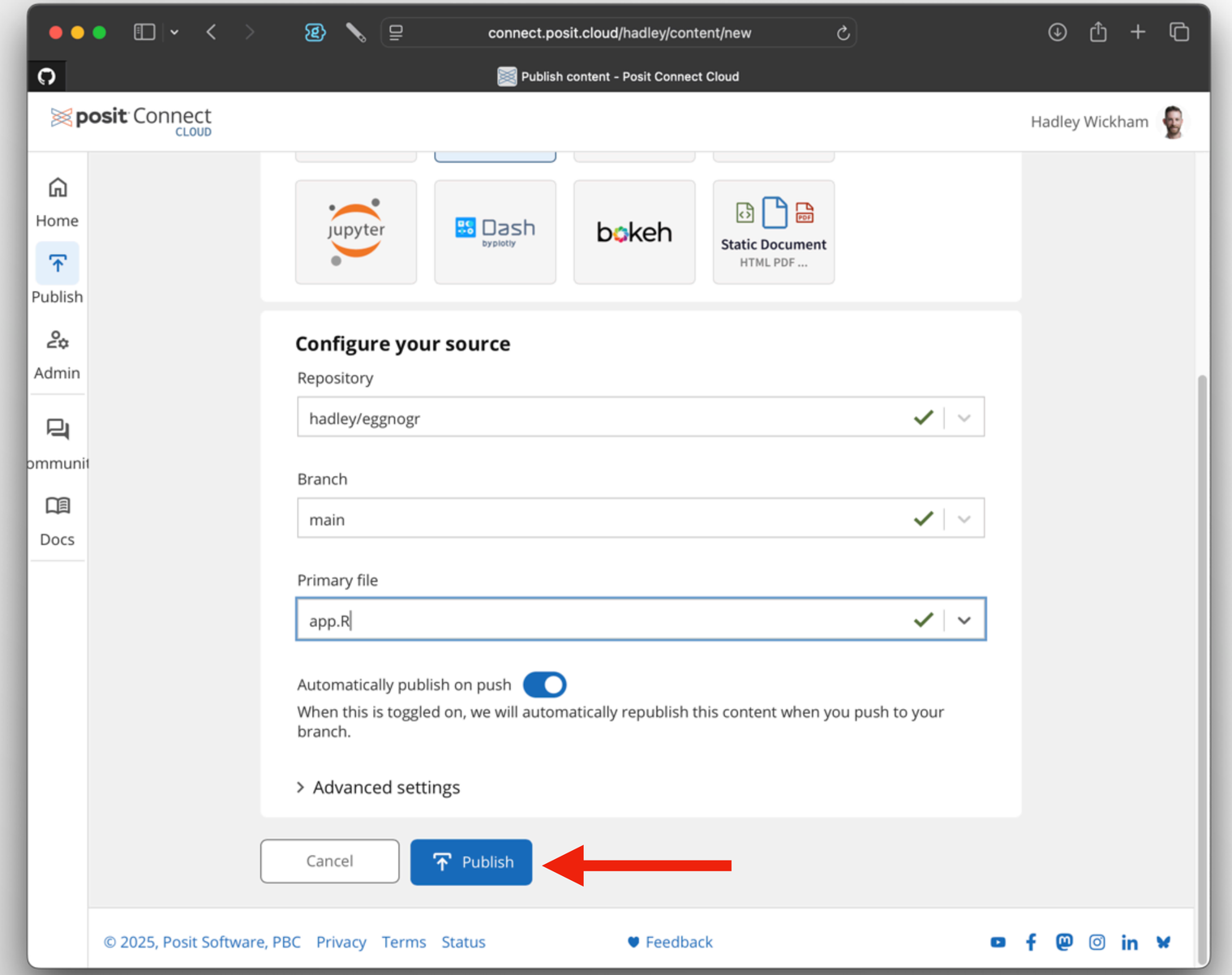
- https://github.com/hadley/eggnogr
- https://github.com/hadley/madlib

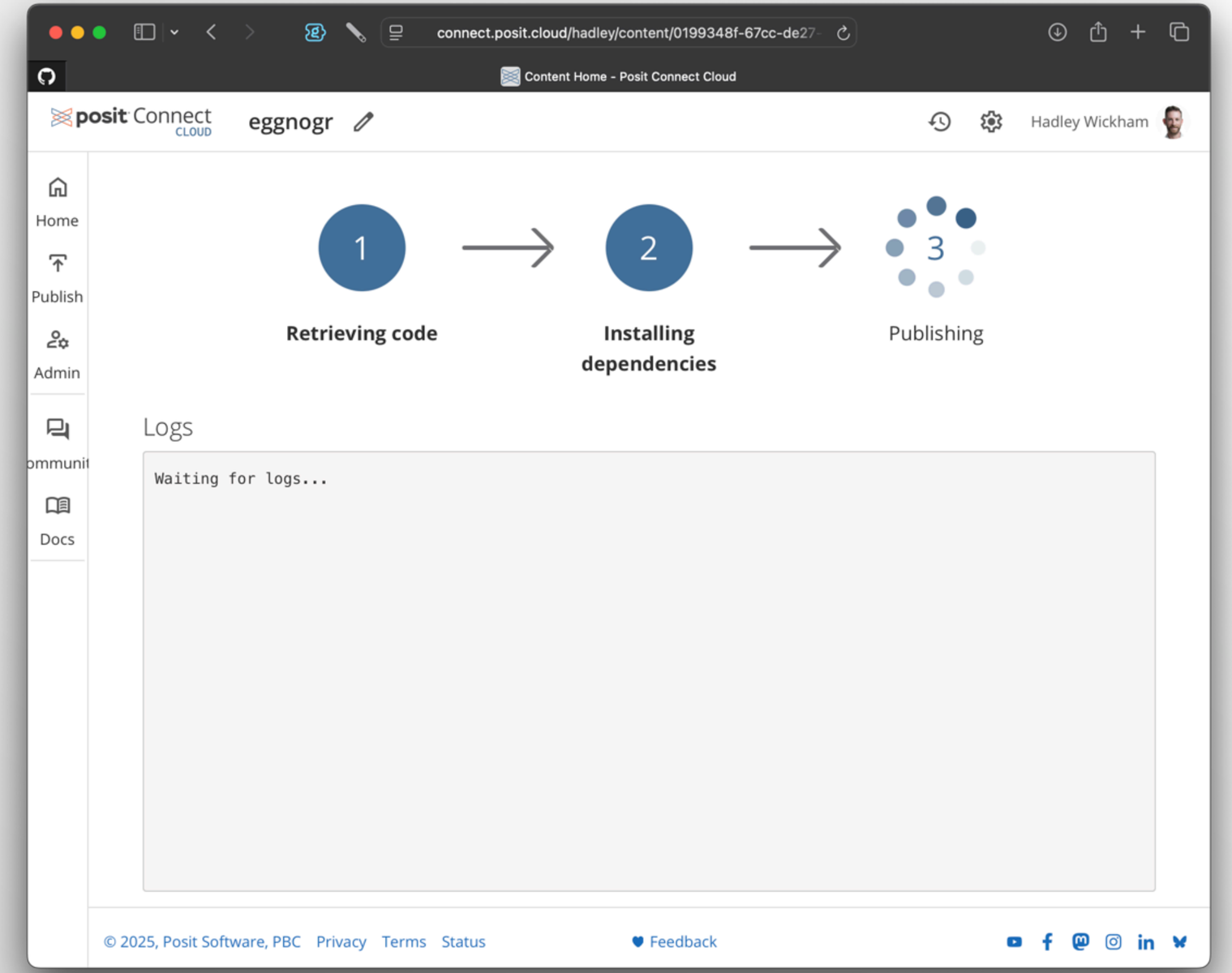


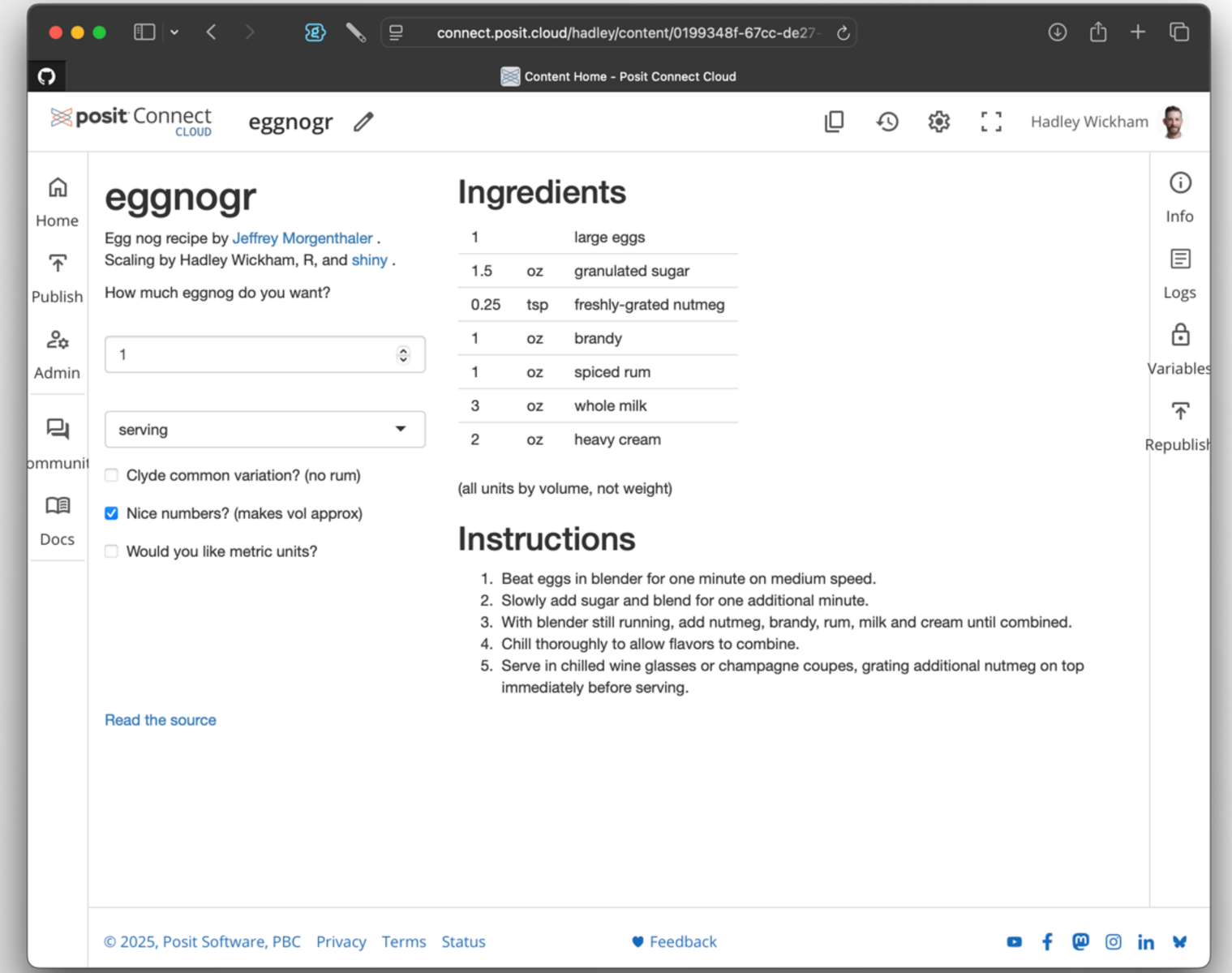
https://connect.posit.cloud/


connect.posit.cloud/hadley
Hadley Wickham - Posit Connect Cloud
posit Connect CLOUD
Hadley Wickham
Home
Publish
Admin
ommunit
Docs
Hadley Wickham
Upgrade Plan
Publish
Search on title, description, or repository
Filter by: Framework
12 results
Sales Data Visualization
product-sales
Public 99d
cocktails-demo
Public 98d
available-work
Startup Error


connect.posit.cloud/hadley/content/new
Publish content - Posit Connect Cloud
posit Connect CLOUD
Hadley Wickham
Home
Publish
Admin
ommunit
Docs
Publish content
Build your own
Use an example
Select your framework
quarto
Shiny
rmarkdown
Streamlit
jupyter
Dash by plotly
bokeh
Static Document
HTML PDF ...
Cancel
© 2025, Posit Software, PBC
Privacy
Terms
Status
Feedback


connect.posit.cloud/hadley/content/new
Publish content - Posit Connect Cloud
posit Connect CLOUD
Hadley Wickham
Home
Publish
Admin
ommunit
Docs
jupyter
Dash by plotly
bokeh
Static Document
HTML PDF ...
Configure your source
Repository
hadley/eggnogr
Branch
main
Primary file
app.R
Automatically publish on push
When this is toggled on, we will automatically republish this content when you push to your branch.
Advanced settings
Cancel
Publish
© 2025, Posit Software, PBC
Privacy
Terms
Status
Feedback


connect.posit.cloud/hadley/content/0199348f-67cc-de27-
Content Home - Posit Connect Cloud
posit Connect CLOUD
eggnogr
Hadley Wickham
Home
Publish
Admin
ommunit
Docs
1
2
3
Retrieving code
Installing dependencies
Publishing
Logs
Waiting for logs...
© 2025, Posit Software, PBC
Privacy
Terms
Status
Feedback


connect.posit.cloud/hadley/content/0199348f-67cc-de27-
Content Home - Posit Connect Cloud
posit Connect CLOUD
eggnogr
Hadley Wickham
Home
Publish
Admin
ommunit
Docs
Info
Logs
Variables
Republish
eggnogr
Egg nog recipe by Jeffrey Morgenthaler .
Scaling by Hadley Wickham, R, and shiny .
How much eggnog do you want?
1
serving
Clyde common variation? (no rum)
Nice numbers? (makes vol approx)
Would you like metric units?
Read the source
Ingredients
1 large eggs
1.5 oz granulated sugar
0.25 tsp freshly-grated nutmeg
1 oz brandy
1 oz spiced rum
3 oz whole milk
2 oz heavy cream
(all units by volume, not weight)
Instructions
1. Beat eggs in blender for one minute on medium speed.
2. Slowly add sugar and blend for one additional minute.
3. With blender still running, add nutmeg, brandy, rum, milk and cream until combined.
4. Chill thoroughly to allow flavors to combine.
5. Serve in chilled wine glasses or champagne coupes, grating additional nutmeg on top immediately before serving.
© 2025, Posit Software, PBC Privacy Terms Status
Feedback

Your turn

- Create a new madlibs project.
- use_git(); use_github().
- What packages do you need?
- Copy in 1-madlibs.R and name it **app.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.
- 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