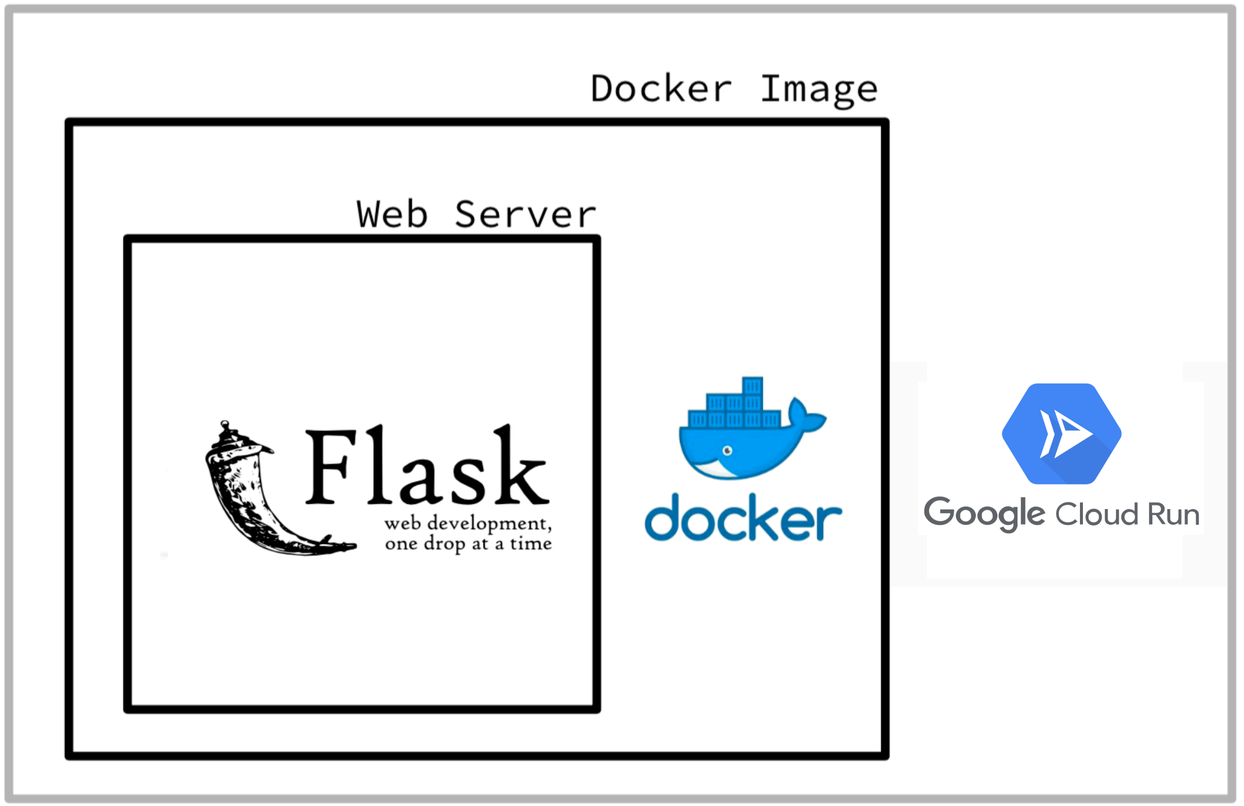
What did we build? (week-7)



As we noted last week, those of you with Apple m1 chip resulted in an error when deploying the built image to Google Cloud Run

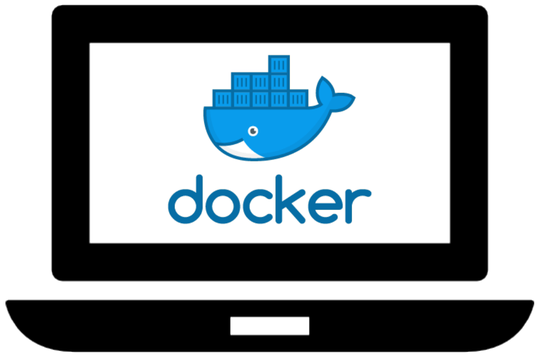
Many of you were able to find the solution to force the platform build (then you aren't using docker compose)

```
docker buildx create --use
docker buildx build --platform linux/amd64 -t docker.io/
setaranusratty/plumber-app:latest . --push
```

Or modifying the docker-compose.yml with right above the ***build*** line

```
platform: "linux/amd64"
```

Deploying a Shiny App 3 Different Ways

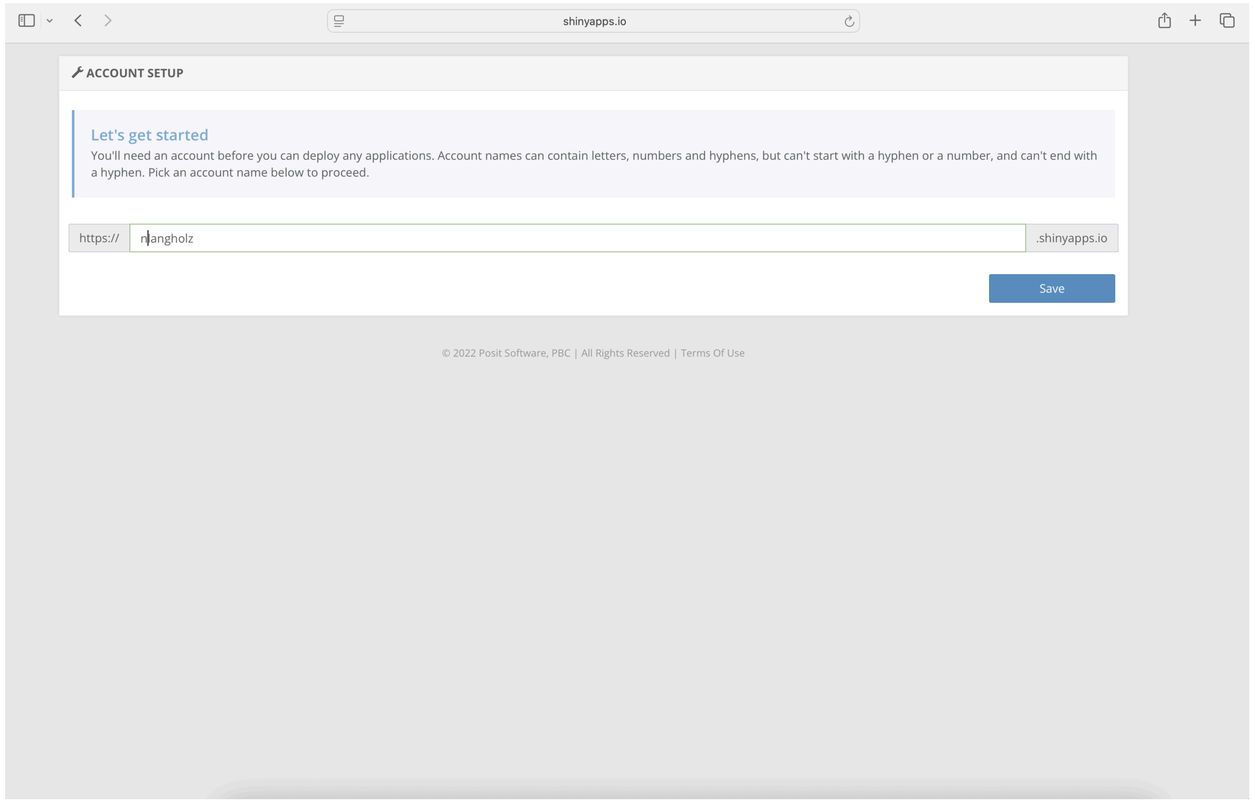


Docker, locally

shinyapps.io

GCR docker

Create an account and use the provided R script in GitHub repo to deploy to shinyapps.io




```
> rsconnect::deployApp(appDir = 'docker/',appName="clusters")
── Preparing for deployment ──────────────────────────────────────────────────
✔ Deploying "clusters" using "server: shinyapps.io / username: nlangholz"
ℹ Creating application on server...
✔ Created application with id 14794727
ℹ Bundling 4 files: 'docker-compose.yml', 'Dockerfile', 'server.R', and 'ui.R'
ℹ Capturing R dependencies
The following required packages are not installed:
- bslib
- httr
- shiny
Packages must first be installed before renv can snapshot them.
Use `renv::dependencies()` to see where this package is used in your project.

What do you want to do?

1: Snapshot, just using the currently installed packages.
2: Install the packages, then snapshot.
3: Cancel, and resolve the situation on your own.

Selection: 2
✔ Found 35 dependencies
✔ Created 22,727b bundle
ℹ Uploading bundle...
✔ Uploaded bundle with id 10315270
── Deploying to server ───────────────────────────────────────────────────────
Waiting for task: 1546369701
  building: Processing bundle: 10315270
  building: Building image: 12723343
  building: Installing packages
  building: Installing files
  building: Pushing image: 12723343
  rollforward: Activating new instances
  success: Stopping old instances
── Deployment complete ───────────────────────────────────────────────────────
✔ Successfully deployed to <https://nlangholz.shinyapps.io/clusters/>
```

Iris Modeling

X Variable

Sepal.Length

Y Variable

Sepal.Width

Cluster count

3

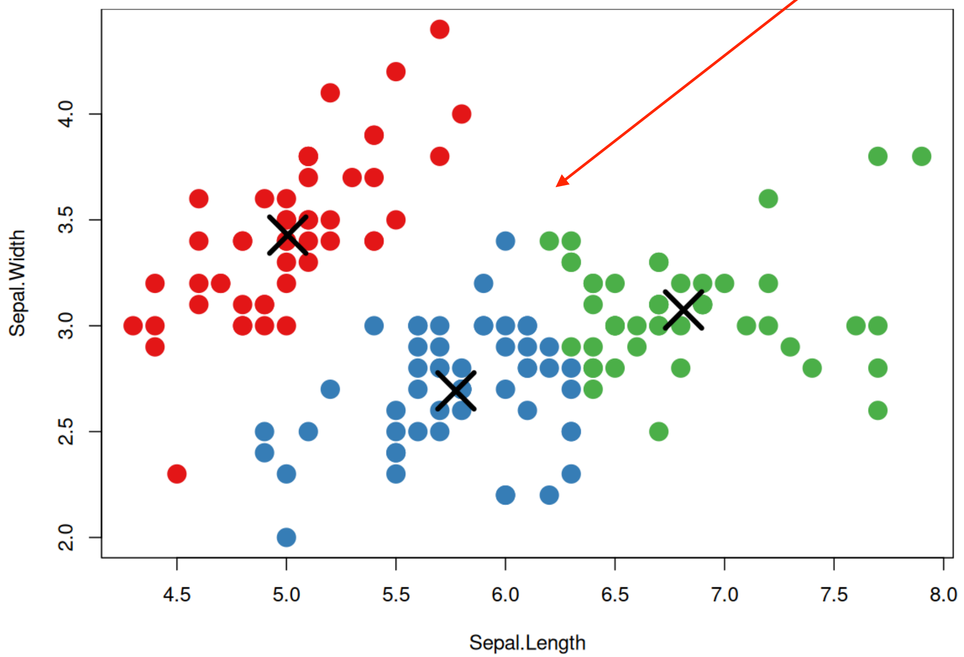
Pick a Sepal Length

5

Pick a Petal Width

6

Iris k-means clustering

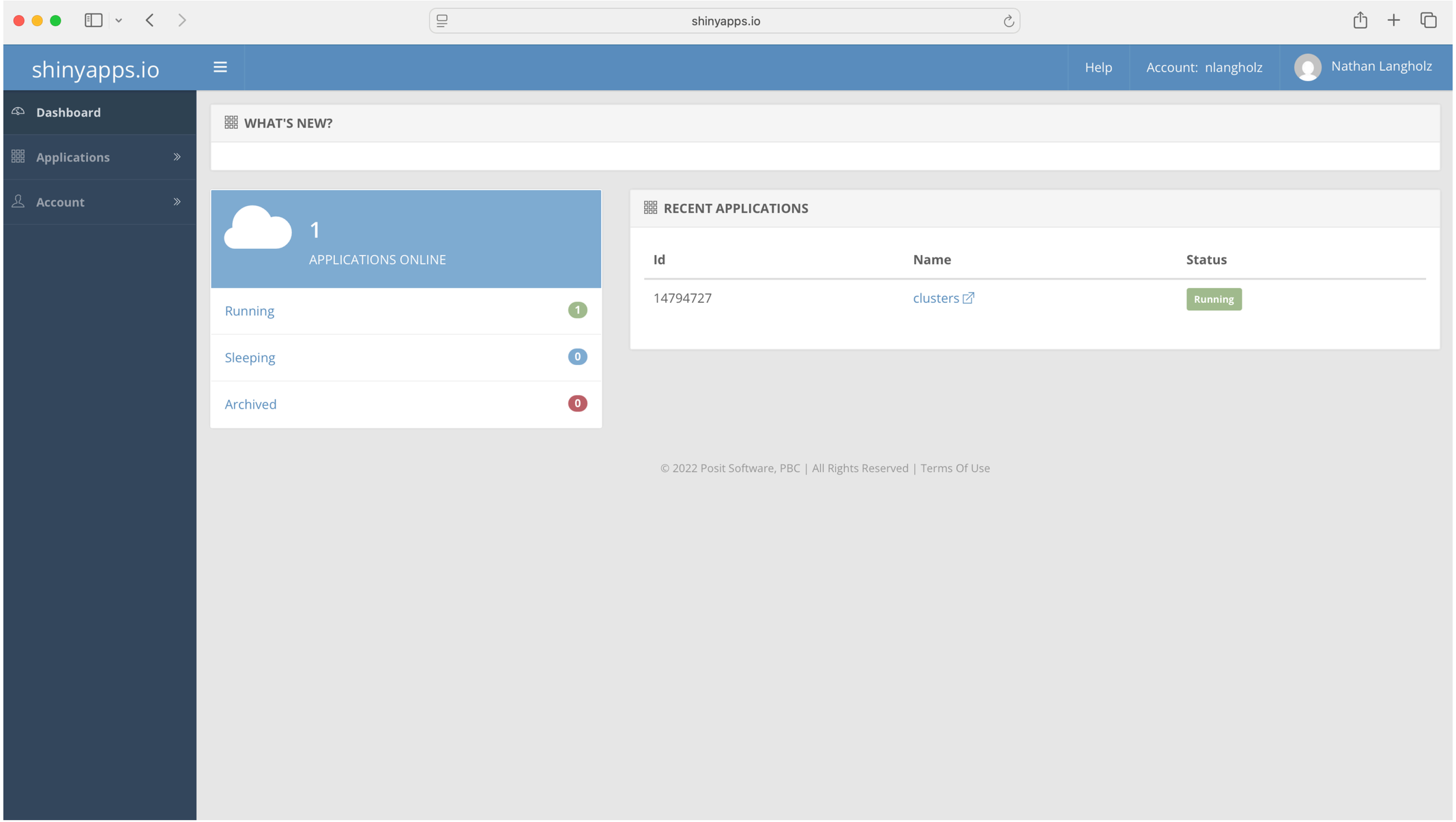


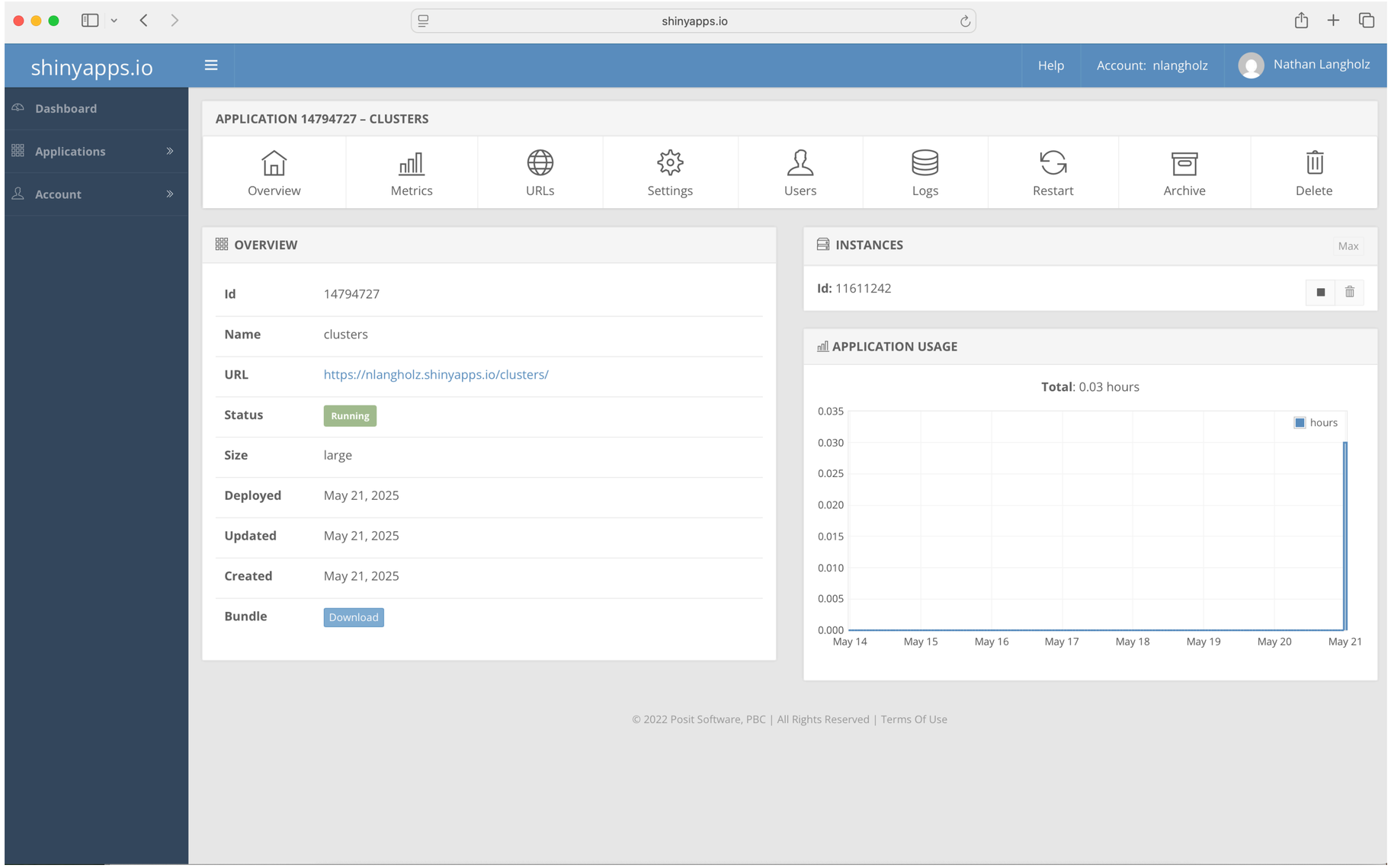
The Predicted Petal Length for your selected Sepal Length and Petal Width is:

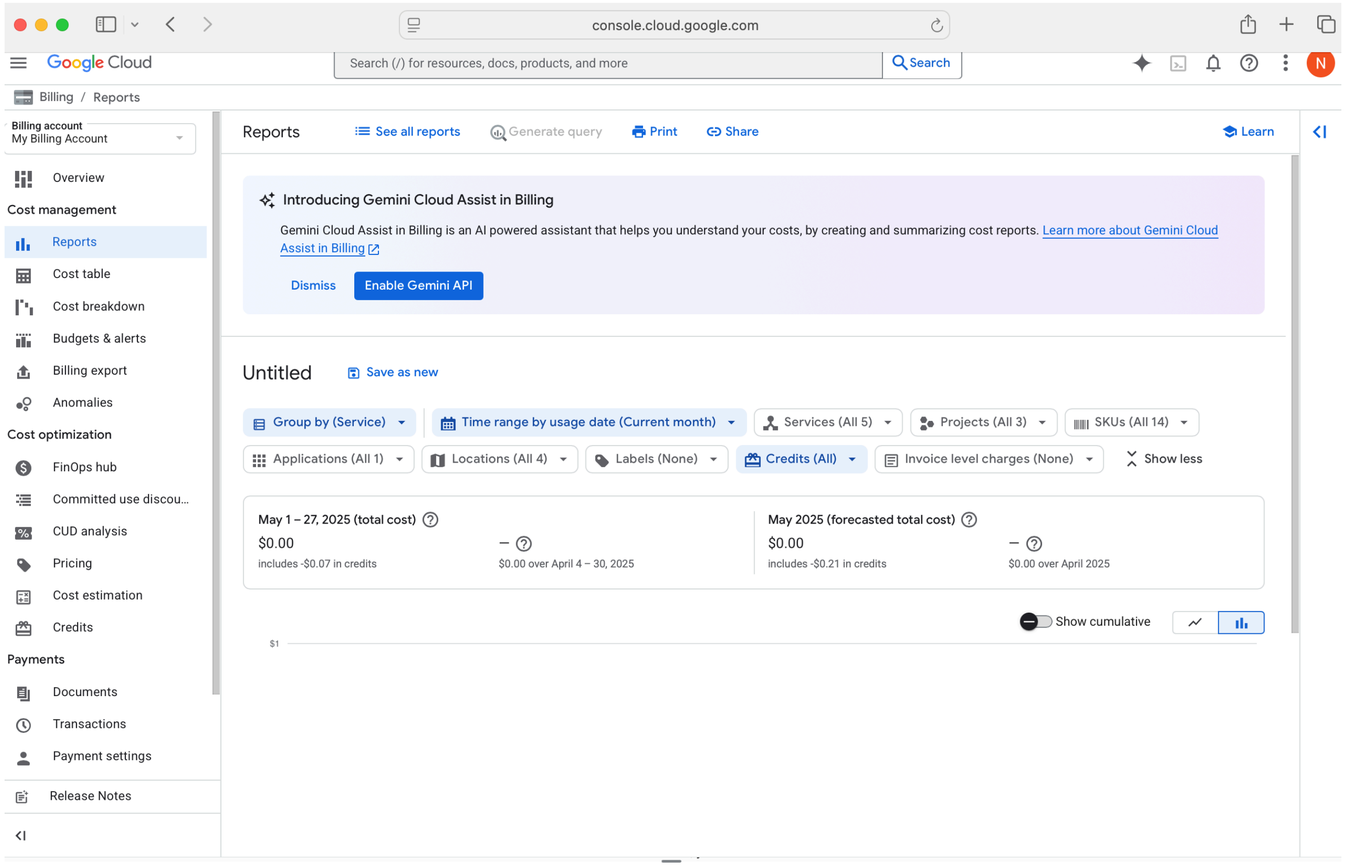
11.6928

K-means model directly embedded in app

GCR api model output; separate service that could've been trained in R or python


shinyapps.io
shinyapps.io
Help
Account: nlangholz
Nathan Langholz
Dashboard
Applications
Account
WHAT'S NEW?
1
APPLICATIONS ONLINE
Running 1
Sleeping 0
Archived 0
RECENT APPLICATIONS
Id
Name
Status
14794727
clusters
Running
© 2022 Posit Software, PBC | All Rights Reserved | Terms Of Use


shinyapps.io
shinyapps.io
Help
Account: nlangholz
Nathan Langholz
Dashboard
Applications
Account
APPLICATION 14794727 – CLUSTERS
Overview
Metrics
URLs
Settings
Users
Logs
Restart
Archive
Delete
OVERVIEW
Id 14794727
Name clusters
URL https://nlangholz.shinyapps.io/clusters/
Status Running
Size large
Deployed May 21, 2025
Updated May 21, 2025
Created May 21, 2025
Bundle Download
INSTANCES
Max
Id: 11611242
APPLICATION USAGE
Total: 0.03 hours
hours
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
May 14
May 15
May 16
May 17
May 18
May 19
May 20
May 21
© 2022 Posit Software, PBC | All Rights Reserved | Terms Of Use



An aside

So far I don’t see any billing charges in my Google cloud account; I can’t find if my images have been copied to gcr

Locally, in terminal

Git clone or zip download my **streamlit-kmeans-418** —repo (not in your forked course repo!)

change into that directory and if you have python and streamlit installed on your machine you could simply run

```
streamlit run app.py
```

Or since there is also a Dockerfile (I don’t have any python on this machine)

Docker on Local Machine

Git clone or zip download my **streamlit-kmeans-418** (not in your forked course repo!)

Change your directory into the folder.

Then (remember include -d if you want it detached an your prompt back):

```
docker compose up -d
```

Now check http://localhost:8080

Deploy to Google Cloud Run.

Deploying to cloud run is the same as when we did with the models and the shiny apps.

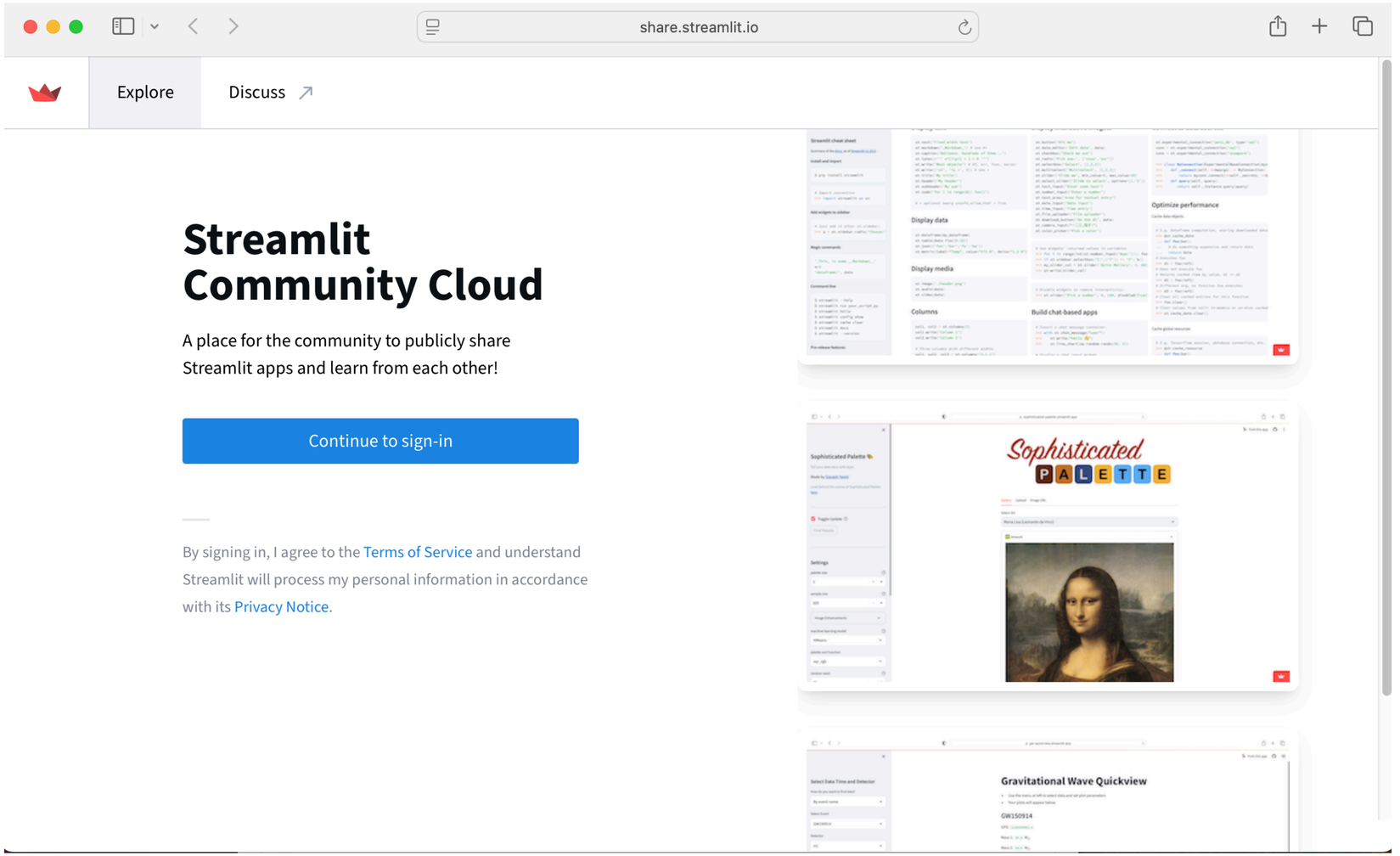
Once you've build your image push to your dockerhub registry and then Create a service. In the Dockerfile you'll notice, I set the port to 8080 again (streamlit standard is 8051) so you won't have to change that in your service

Streamlit

A faster way to build and share data apps

Turn your data scripts into shareable web apps in minutes.
All in pure Python. No front-end experience required.

Straight from their website



Also, has free deployment and also need to create an account. Lots of examples.

https://share.streamlit.io/explore?sort=most+viewed&category=favorites

So what’s the difference between Streamlit and Shiny?

Largely most people say Streamlit shines in rapid development and ease of use, ideal for quick prototypes and data exploration tasks.

In comparison, Shiny is more of extensive framework for complex, feature-rich data apps, with PyShiny extending these capabilities to Python users.

Both platforms come with their unique strengths and limitations and deciding between the two depends on specific project requirements

High level comparison of Streamlit, RShiny, PyShiny

Criteria	Streamlit	RShiny	PyShiny
Language Ecosystem	Python-native	R-native	Python-native
Development Speed	Rapid Prototyping	Suited for both medium and long-term projects	Suited for both medium and long-term projects
Ease of Use	High (few lines of code)	Moderate to High (depends on customization)	Moderate to High (depends on customization)
Customization	Limited but improving	Advanced, pixel-level	Advanced, pixel-level
Community Support	Strong and growing (27k+ stars on GitHub)	Well-established and mature	Emerging, building upon RShiny community
Widgets	Increasing variety but less comprehensive	Wide range of pre-implemented widgets	Base implemented, supports all ipywidgets, more under development
Best For	Quick prototypes, data exploration	Ideal for medium-complex to feature-rich, robust dashboards	Ideal for medium-complex to feature-rich, robust dashboards in Python
Specialized Sectors	Fields requiring quick development cycles	Sectors needing varied complexity and strong reliability	Sectors needing varied complexity and strong reliability in Python
Examples	Proof-of-concept ML models, temporary dashboards	Scientific computing apps, enterprise-grade applications	Scientific computing apps, enterprise-grade applications in Python

Taken from https://www.appsilon.com/post/streamlit-or-shiny-for-life-scientists

posit.co has a good comparison (obviously pro-shiny) at https://shiny.posit.co/py/docs/comp-streamlit.html parts of it below.

Shiny and Streamlit differ in a few key ways

1. Shiny's reactive execution means that elements are minimally re-rendered.

2. You can build large Shiny applications without manually managing application state or caching data.

3. Shiny allows you to easily customize the look and feel of your application.

Shiny allows you to build much more performant and extensible applications than Streamlit. The patterns that you use to build a simple Shiny application are the same ones that you use to build a complex one, and you never need to change your mental model of how the application works. This design will let your application grow along with the scope of your problem, and you can have confidence that the framework has the tools that you need to handle almost any requirement.

posit.co has a good comparison (obviously pro-shiny) at https://shiny.posit.co/py/docs/comp-streamlit.html parts of it below.

Shiny and Streamlit differ in a few key ways

1. Shiny's reactive execution means that elements are minimally re-rendered.

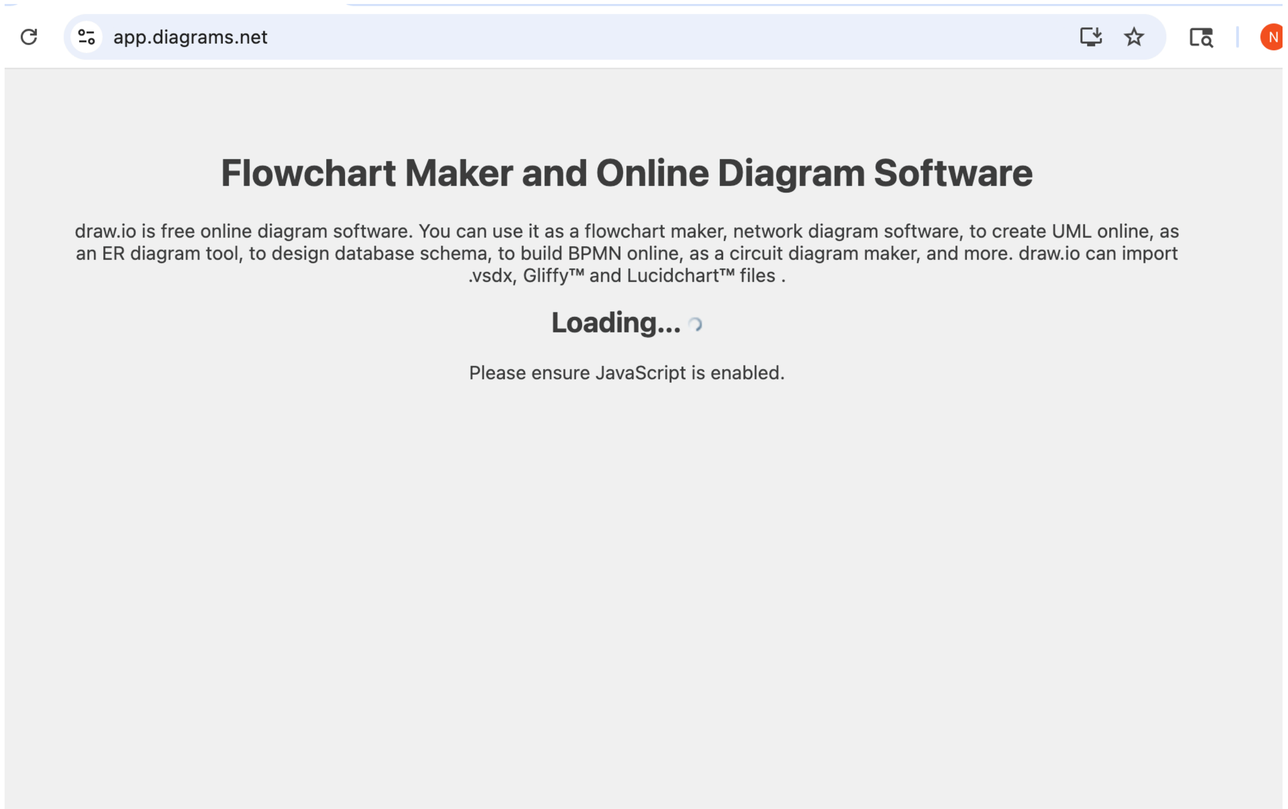
2. You can build large Shiny applications without manually managing application state or caching data.

3. Shiny allows you to easily customize the look and feel of your application.

Shiny allows you to build much more performant and extensible applications than Streamlit. The patterns that you use to build a simple Shiny application are the same ones that you use to build a complex one, and you never need to change your mental model of how the application works. This design will let your application grow along with the scope of your problem, and you can have confidence that the framework has the tools that you need to handle almost any requirement.

draw.io is a nice online tool for architecture diagrams (url actually changed to **app.diagrams.net**)

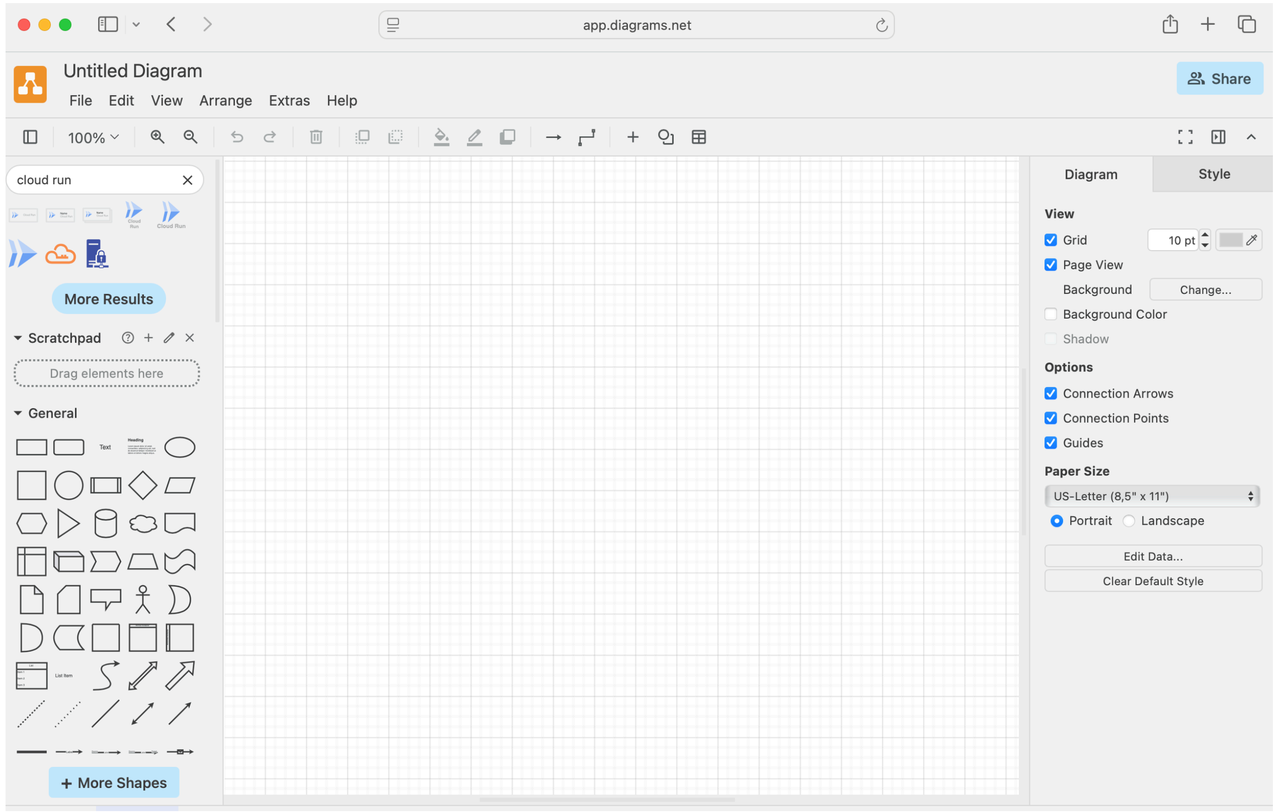
As you begin to design more systems and applications this becomes a helpful visualization, informational, and story telling



A blank canvas to create your architecture diagram; there are also a number templates that have been created or other people have created

https://github.com/kennethleungty/Neural-Network-Architecture-Diagrams

There is also a suite of integrated icons from large cloud providers that you can easily import and well as just importing your own

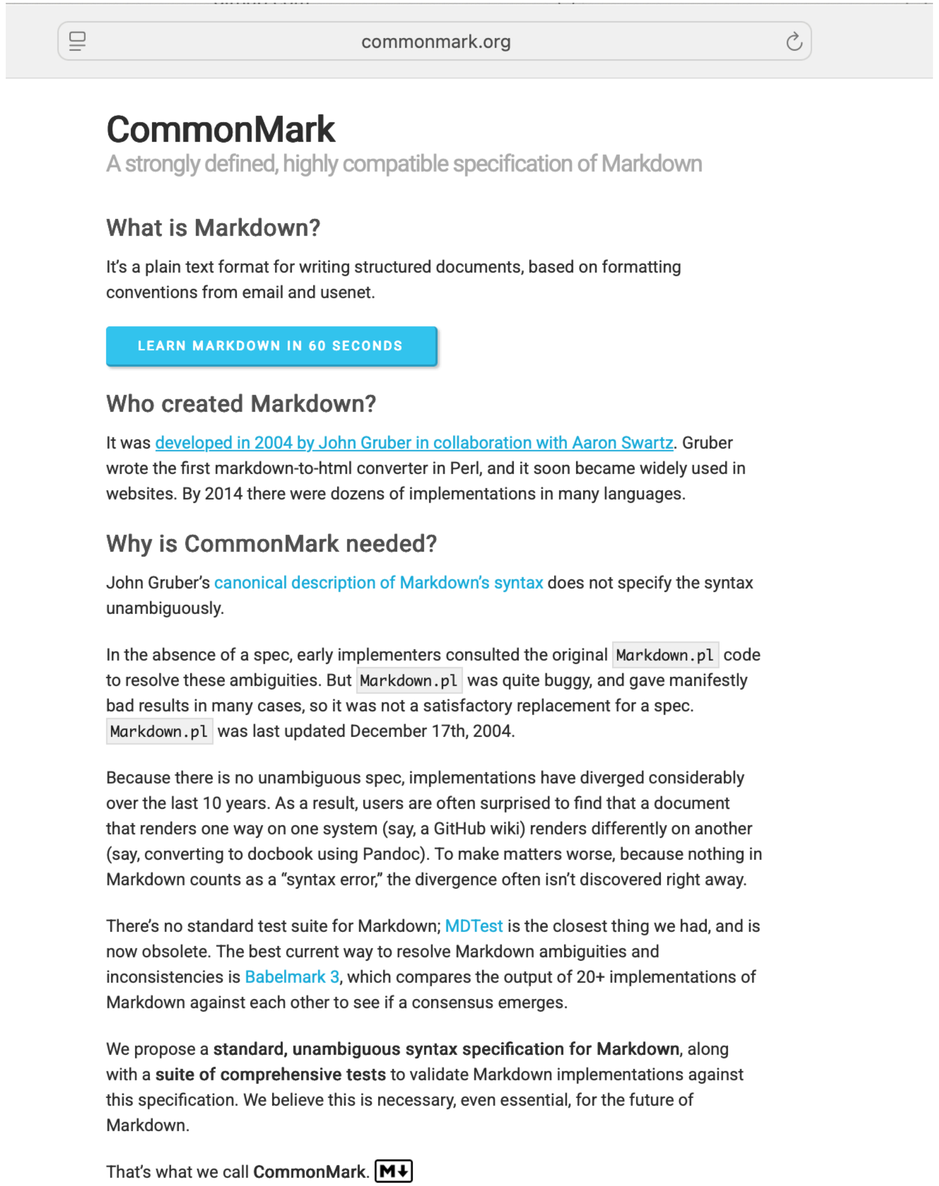


One last tool, I guess…Markdown.

We've already seen this in action in our Github README's as well as versions of Rmarkdowns.

Worth taking a look at the 60-second tutorial and the 10-minute one as well.

https://commonmark.org/



This ends our survey of variety of different Data Science Tools from data collection all the way through various deployments. We saw quite a few different things throughout the course!




Where to go from here?

Spend more time intergrating AI into workflows and building AI Agents and applications with AI platforms and leveraging GPUs, etc

Where to go from here?

Start to understand load balancing, security, and other deployment considerations as well as model monitoring and governance at implementation level.

Some final general advice

1. Practice Curiosity Driven Data Science

2. Do public work

3. Build (y)our community

Curiosity Driven Data Science

1. Don’t learn every tool but general frameworks and how to solve problems

2. Ask questions and dig into things you find interesting

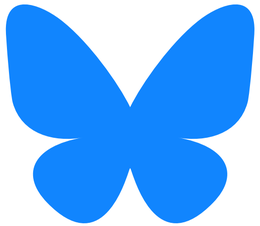
3. Story telling and insight generation with iterative feedback loops

Do Public Work

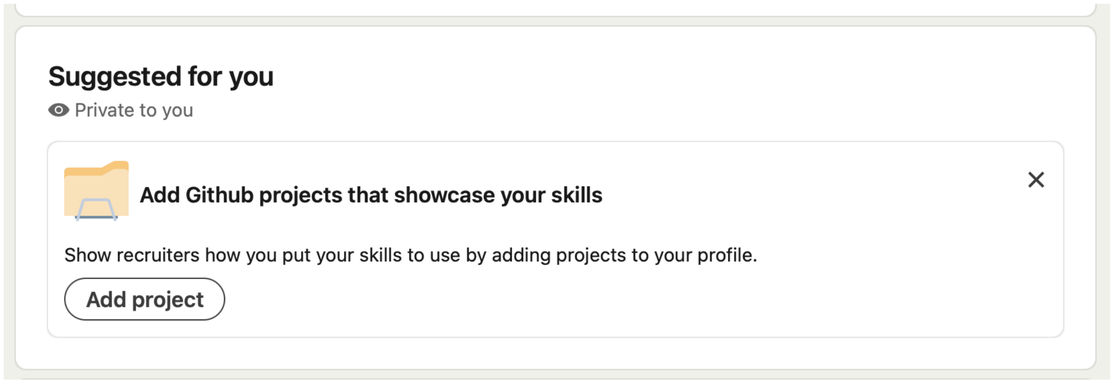
1. Speak at events

2. Contribute to open source or a feature to a GitHub repo

3. Build eminence through blogs, other public projects, your own website



This section showed up recently on my LinkedIn profile



Build (y)our community

1. Attend community events and bigger conferences

2. Keep supporting MASDS and UCLA

3. Network through LinkedIn, Github, Bluesky(?)

Add all of us and each other on LinkedIn

1. Nate Langholz - https://www.linkedin.com/in/nate-langholz/

2. Jeremy Weidner - https://www.linkedin.com/in/jeremy-weidner/

3. Alex Steiner - https://www.linkedin.com/in/alex-steiner-9868575/