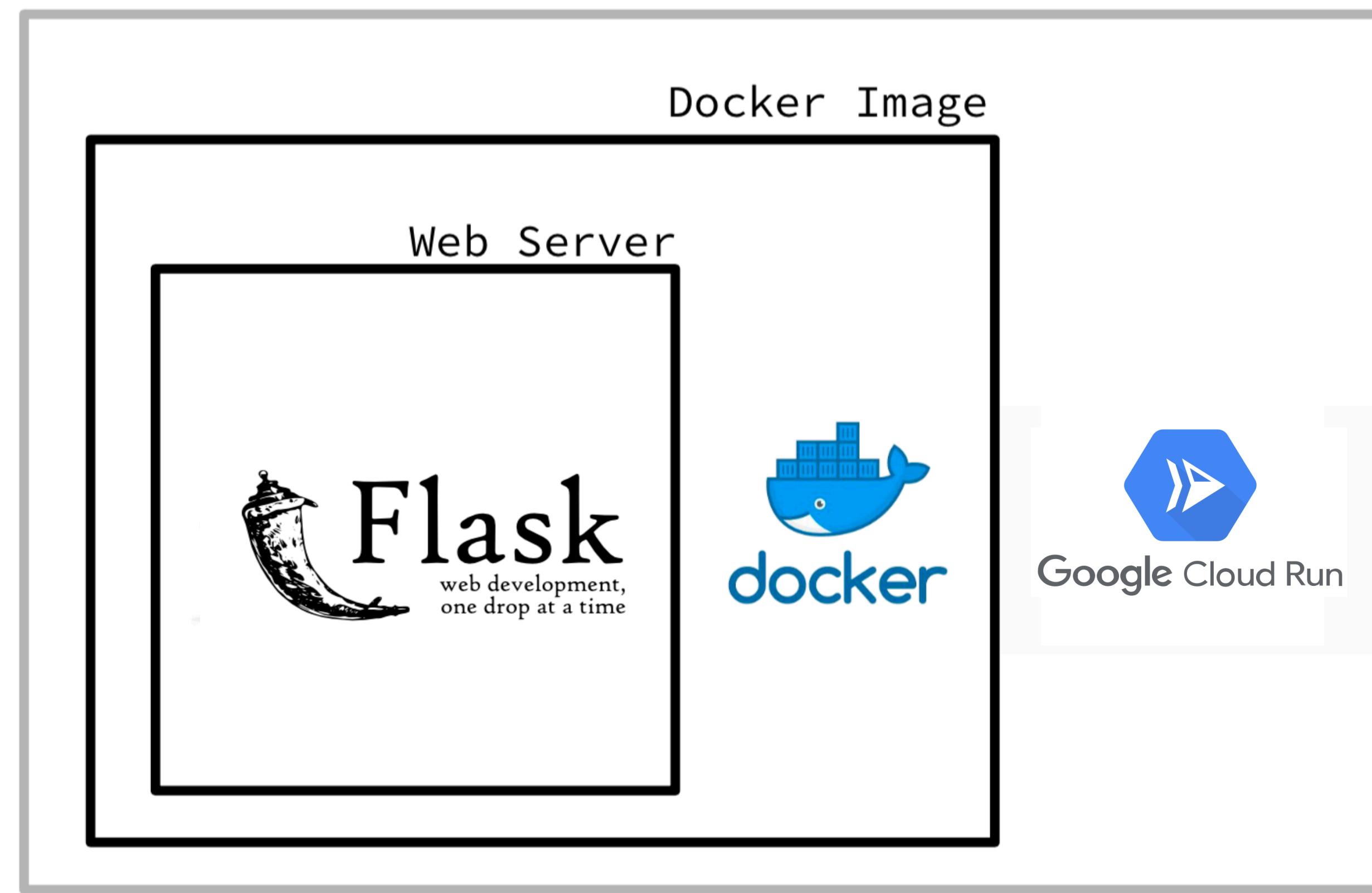


## What did we build? (week-7)



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

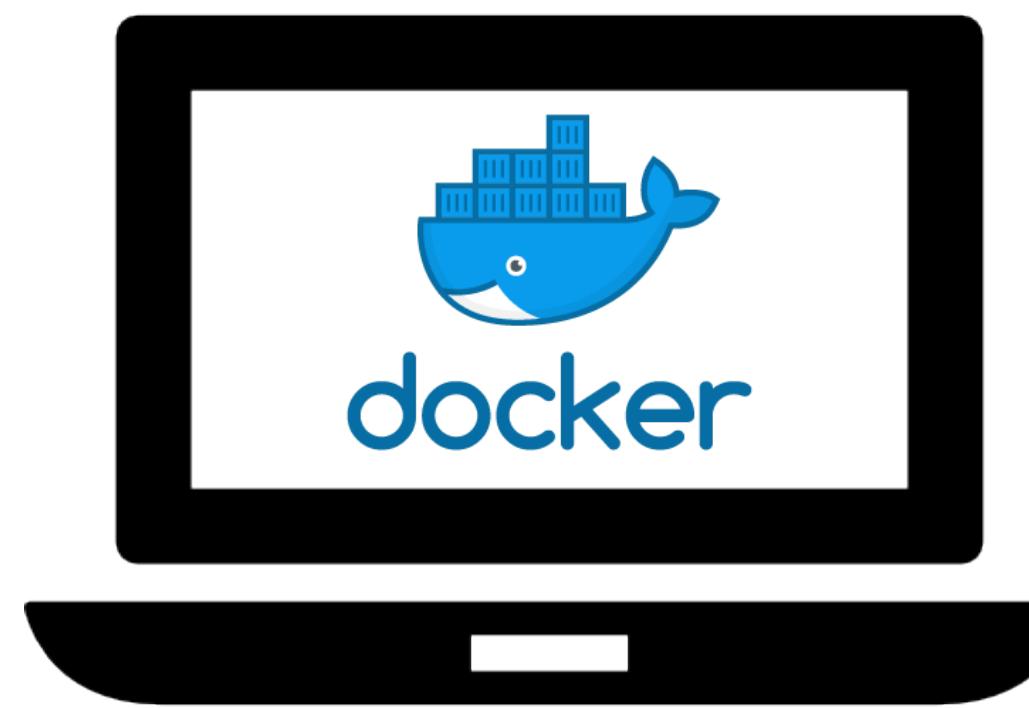
Many of you were able to find the solution to force the platform build (then you aren't using docker compose)

```
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



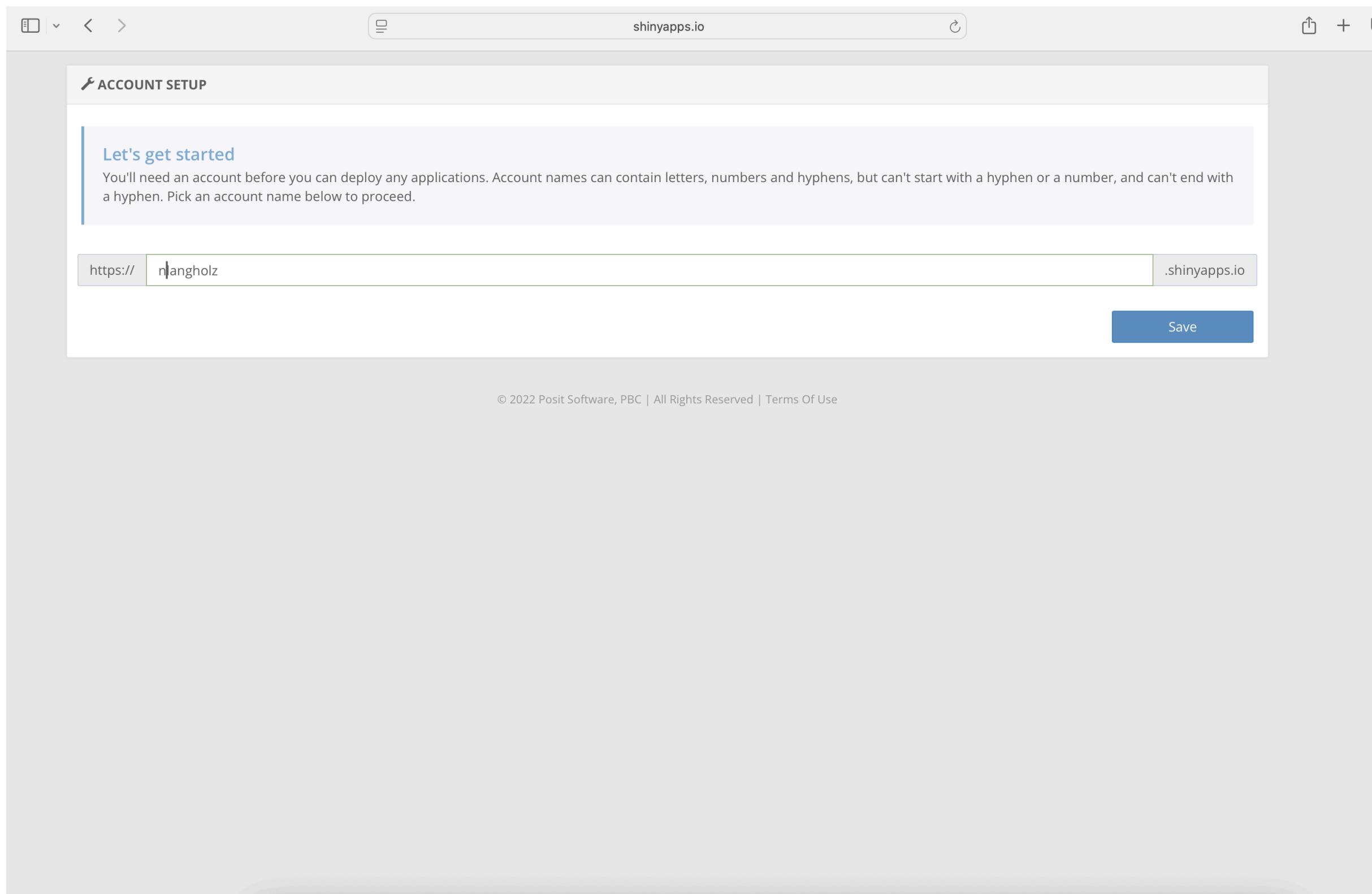
shinyapps.io



Google Cloud Run

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

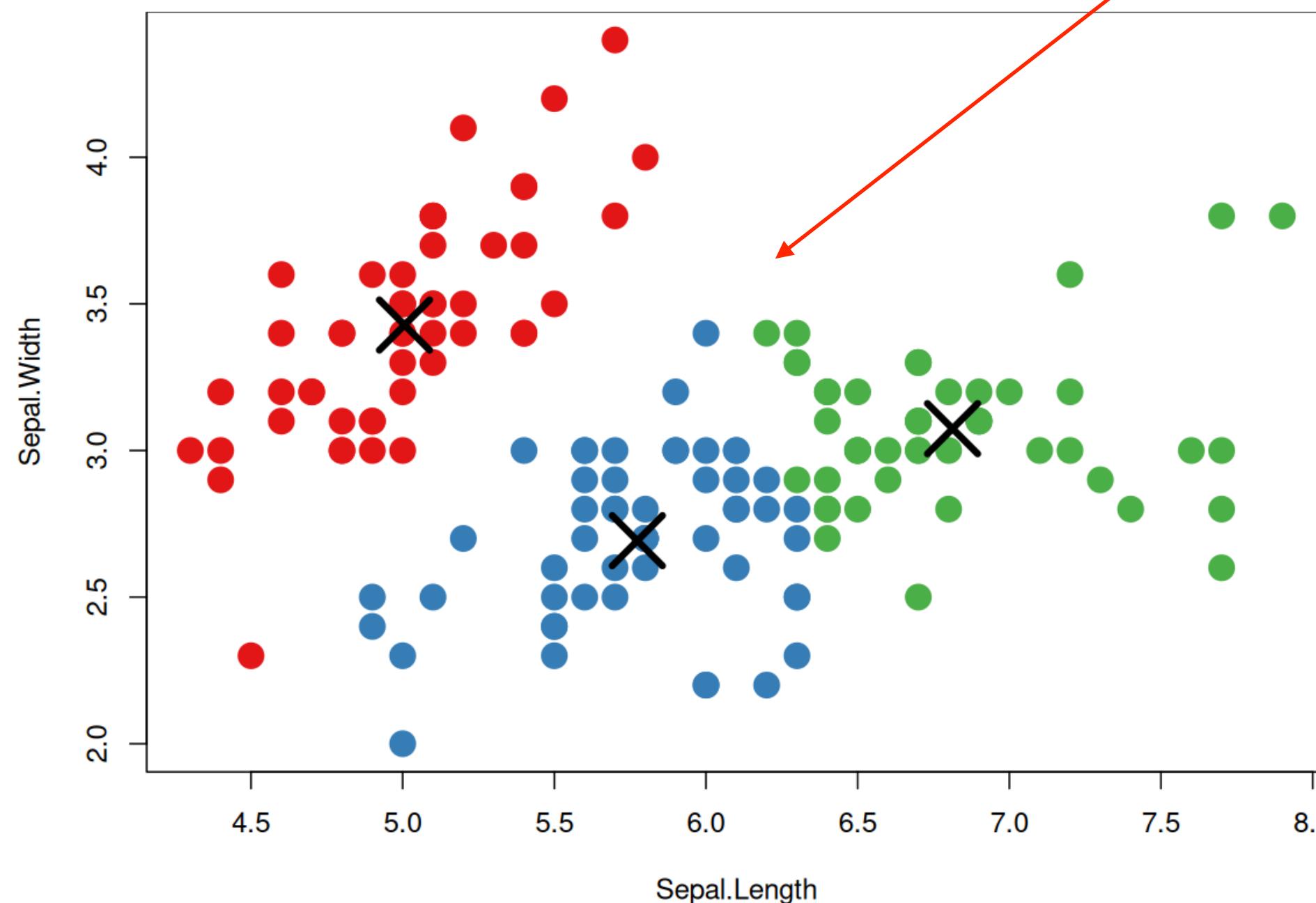
Pick a Sepal Length

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

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

shinyapps.io

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

APPLICATION 14794727 - CLUSTERS

OVERVIEW

<b>Id</b>	14794727
<b>Name</b>	clusters
<b>URL</b>	<a href="https://nlangholz.shinyapps.io/clusters/">https://nlangholz.shinyapps.io/clusters/</a>
<b>Status</b>	Running
<b>Size</b>	large
<b>Deployed</b>	May 21, 2025
<b>Updated</b>	May 21, 2025
<b>Created</b>	May 21, 2025
<b>Bundle</b>	<a href="#">Download</a>

INSTANCES

APPLICATION USAGE

Total: 0.03 hours

© 2022 Posit Software, PBC | All Rights Reserved | Terms Of Use

The screenshot shows the Google Cloud Billing / Reports interface. The left sidebar includes sections for Billing account, Overview, Cost management (Reports selected), Cost table, Cost breakdown, Budgets & alerts, Billing export, Anomalies, Cost optimization (FinOps hub, Committed use discounts, CUD analysis, Pricing, Cost estimation, Credits), Payments (Documents, Transactions, Payment settings, Release Notes), and Gemini Cloud Assist in Billing. The main area displays an 'Untitled' report titled 'May 1 – 27, 2025 (total cost)'. It shows a total cost of \$0.00, including -\$0.07 in credits over April 4 – 30, 2025. A forecasted total cost for May 2025 is also shown at \$0.00, including -\$0.21 in credits over April 2025. The report is grouped by Service, Time range by usage date (Current month), Services (All 5), Projects (All 3), SKUs (All 14), Applications (All 1), Locations (All 4), Labels (None), Credits (All), and Invoice level charges (None). A 'Gemini Cloud Assist in Billing' notification is present, encouraging users to learn more about the AI-powered cost reporting feature.

## 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



# Streamlit

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

The screenshot shows the Streamlit Community Cloud interface at [share.streamlit.io](https://share.streamlit.io). At the top, there's a navigation bar with a crown icon, 'Explore', and 'Discuss'. Below the header, the main title 'Streamlit Community Cloud' is displayed in large, bold, dark font. A subtitle below it reads: 'A place for the community to publicly share Streamlit apps and learn from each other!'. A blue button labeled 'Continue to sign-in' is present. To the right of the title, there are three examples of Streamlit dashboards:

- Streamlit cheat sheet**: A dashboard containing a large amount of Streamlit code snippets for various components like charts, tables, and media.
- Sophisticated PALETTE**: A dashboard featuring a sidebar for settings and a main area displaying the Mona Lisa painting.
- Gravitational Wave Quickview**: A dashboard showing a plot related to GW150914 gravitational wave event.

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
<b>Language</b>	Python-native	R-native	Python-native
<b>Ecosystem</b>			
<b>Development</b>	Rapid Prototyping	Suited for both medium and long-term projects	Suited for both medium and long-term projects
<b>Speed</b>			
<b>Ease of Use</b>	High (few lines of code)	Moderate to High (depends on customization)	Moderate to High (depends on customization)
<b>Customization</b>	Limited but improving	Advanced, pixel-level	Advanced, pixel-level
<b>Community</b>	Strong and growing	Well-established and mature	Emerging, building upon RShiny community
<b>Support</b>	(27k+ stars on GitHub)		
<b>Widgets</b>	Increasing variety but less comprehensive	Wide range of pre-implemented widgets	Base implemented, supports all ipywidgets, more under development
<b>Best For</b>	Quick prototypes, data exploration	Ideal for medium-complex to feature-rich, robust dashboards	Ideal for medium-complex to feature-rich, robust dashboards in Python
<b>Specialized</b>	Fields requiring quick development cycles	Sectors needing varied complexity and strong reliability	Sectors needing varied complexity and strong reliability in Python
<b>Sectors</b>			
<b>Examples</b>	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.appslion.com/post/streamlit-or-shiny-for-life-scientists>

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

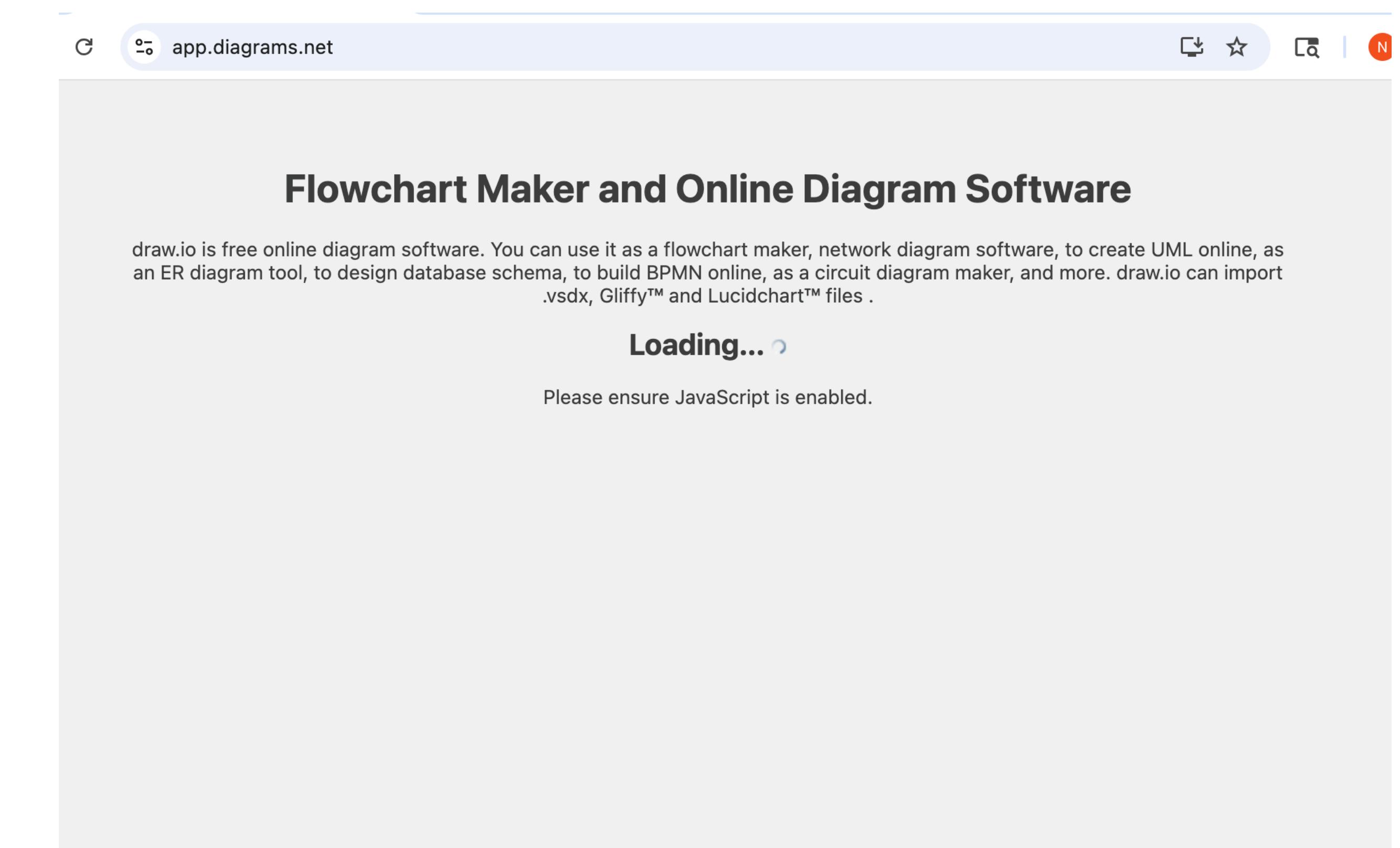
Shiny and Streamlit differ in a few key ways

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**)

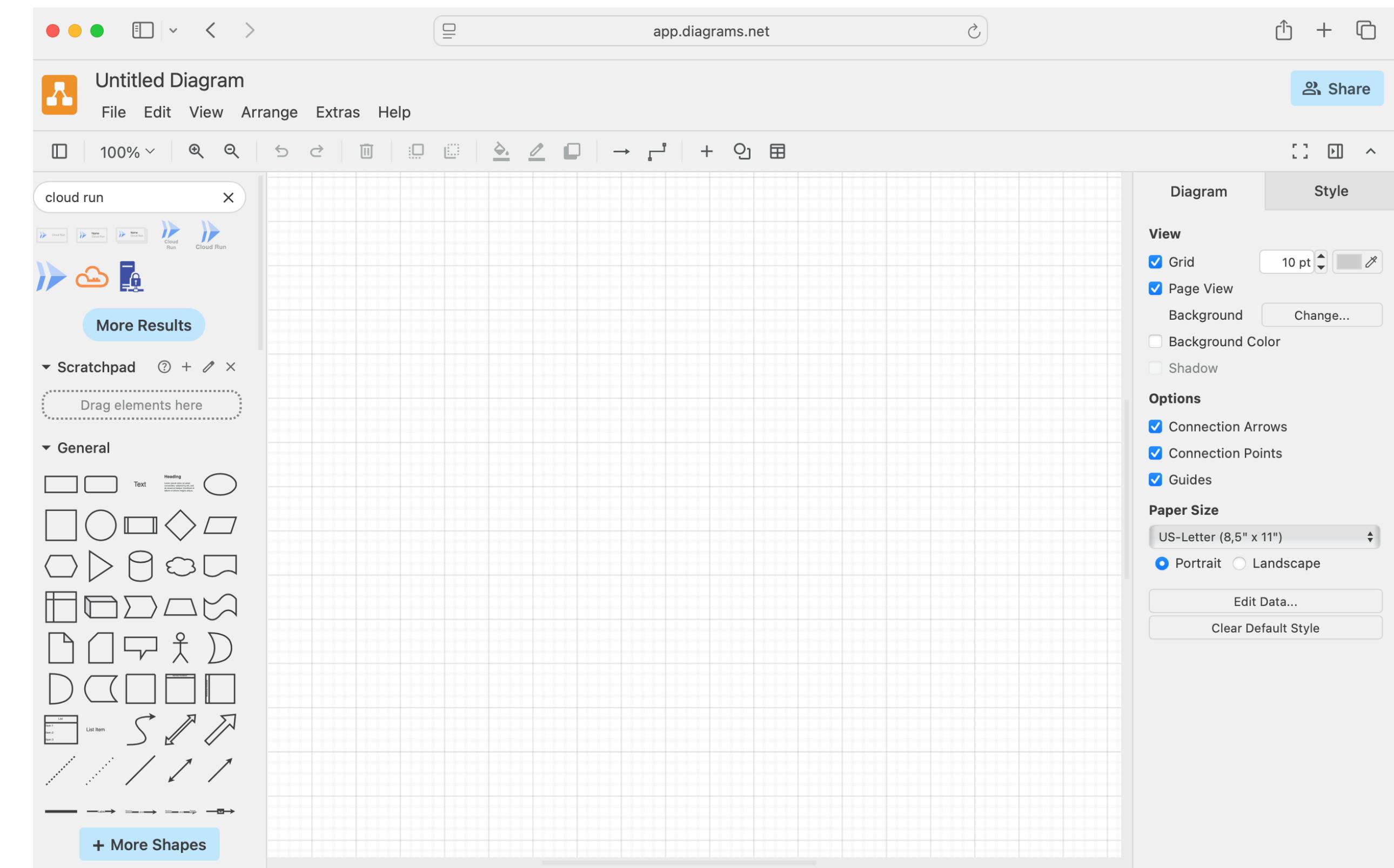
As you begin to design more systems and applications this becomes a helpful visualization, informational, and story telling



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](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

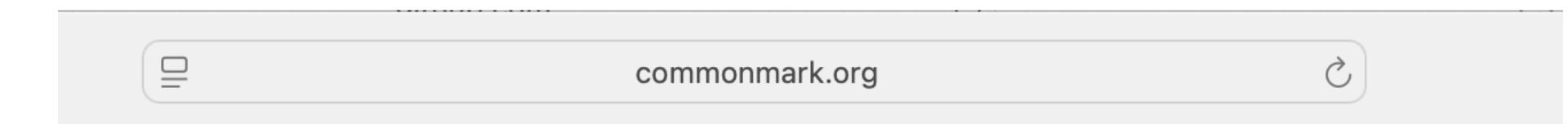


## One last tool, I guess...Markdown.

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

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

<https://commonmark.org/>



### CommonMark

A strongly defined, highly compatible specification of Markdown

#### What is Markdown?

It's a plain text format for writing structured documents, based on formatting conventions from email and usenet.

[LEARN MARKDOWN IN 60 SECONDS](#)

#### Who created Markdown?

It was [developed in 2004 by John Gruber in collaboration with Aaron Swartz](#). Gruber wrote the first markdown-to-html converter in Perl, and it soon became widely used in websites. By 2014 there were dozens of implementations in many languages.

#### Why is CommonMark needed?

John Gruber's [canonical description of Markdown's syntax](#) does not specify the syntax unambiguously.

In the absence of a spec, early implementers consulted the original [Markdown.pl](#) code to resolve these ambiguities. But [Markdown.pl](#) was quite buggy, and gave manifestly bad results in many cases, so it was not a satisfactory replacement for a spec.

[Markdown.pl](#) was last updated December 17th, 2004.

Because there is no unambiguous spec, implementations have diverged considerably over the last 10 years. As a result, users are often surprised to find that a document that renders one way on one system (say, a GitHub wiki) renders differently on another (say, converting to docbook using Pandoc). To make matters worse, because nothing in Markdown counts as a "syntax error," the divergence often isn't discovered right away.

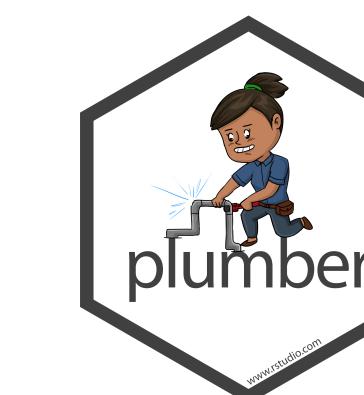
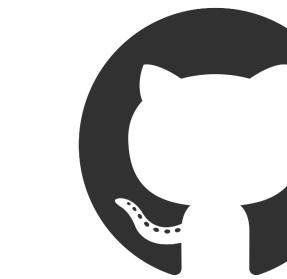
There's no standard test suite for Markdown; [MDTest](#) is the closest thing we had, and is now obsolete. The best current way to resolve Markdown ambiguities and inconsistencies is [Babelmark 3](#), which compares the output of 20+ implementations of Markdown against each other to see if a consensus emerges.

We propose a **standard, unambiguous syntax specification for Markdown**, along with a **suite of comprehensive tests** to validate Markdown implementations against this specification. We believe this is necessary, even essential, for the future of Markdown.

That's what we call **CommonMark**.



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 integrating 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

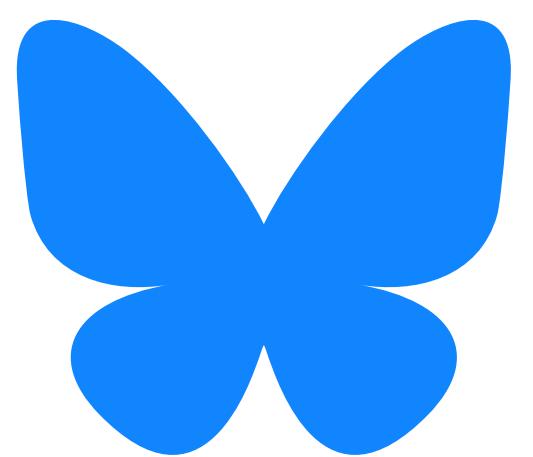
1. Practice Curiosity Driven Data Science
2. Do public work
3. Build (y)our community

## Curiosity Driven Data Science

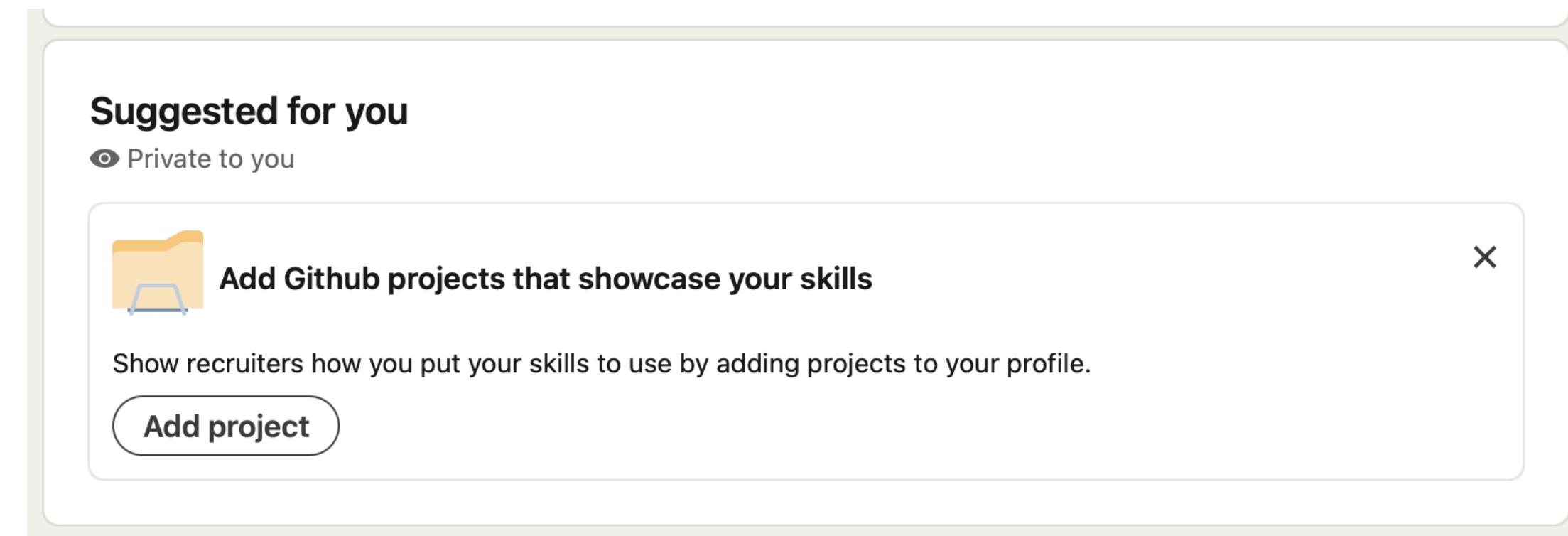
1. Don't learn every tool but general frameworks and how to solve problems
2. Ask questions and dig into things you find interesting
3. Story telling and insight generation with iterative feedback loops

## 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/>