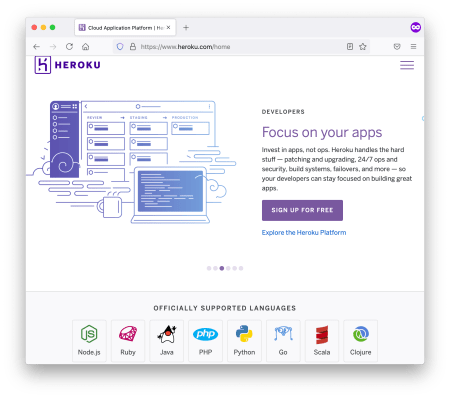
**What is Heroku**

[Heroku](https://www.heroku.com/) is a cloud platform-as-a-service (PaaS) that lets you deploy, run and manage applications. Heroku is also part of the Salesforce Platform, enabling enterprises to store and leverage customer data in Salesforce for full-cycle CRM engagement.

An **application** (app) is the combination of its *source code* and the *dependency description* that determines how to build and run the application. The build mechanism is typically language-specific and is based on so-called [**Buildpacks**](https://devcenter.heroku.com/articles/buildpacks).

The Heroku landing page: R is not officially supported

Because R is not one of the [officially supported languages](https://devcenter.heroku.com/articles/buildpacks#officially-supported-buildpacks) for Heroku, Buildpacks for R are all **community maintained** and might lag (e.g. [this one](https://elements.heroku.com/buildpacks/dpmccabe/heroku-buildpack-r) has R 3.4.3). This [R Buildpack GitHub repository](https://github.com/virtualstaticvoid/heroku-buildpack-r) is quite recent and supports packrat or renv based workflows for Shiny and Plumber.

The main caveat for these R Buildpacks is of course not that these are community maintained. Huge thanks to the maintainers for simplifying our lives and doing the heavy lifting for everyone else! The main caveat in their own words is that:

If any of your R packages dependend on system libraries which aren't [included by Heroku](https://devcenter.heroku.com/articles/stack-packages), such as

libgmp

,

libgomp

,

libgdal

,

libgeos

and

libgsl

, you should use the Heroku [container stack](https://devcenter.heroku.com/categories/deploying-with-docker) together with [heroku-docker-r](https://github.com/virtualstaticvoid/heroku-docker-r) instead. – Maintainers of [heroku-buildpack-r](https://github.com/virtualstaticvoid/heroku-buildpack-r)

What this means is that it is recommended to use a **Docker-based stack** to handle all the finicky **system dependencies**. As you have seen before, a [Docker-based workflow with Shiny](https://hosting.analythium.io/dockerized-shiny-apps-with-dependencies/) gives us an opportunity for local testing before deployment, which might also be very useful.

Heroku is really well suited for a Docker-based workflow because all Heroku applications run in a collection of lightweight [Linux containers called **dynos**](https://devcenter.heroku.com/categories/dynos).

**Prerequisites**

To follow this tutorial, you'll need to sign up for [Heroku](https://www.heroku.com/). It is free, to begin with. Read more about the general [Heroku pricing](https://www.heroku.com/pricing), the [dyno types](https://devcenter.heroku.com/articles/dyno-types), and [free dyno hours](https://devcenter.heroku.com/articles/free-dyno-hours).

[Account verification](https://devcenter.heroku.com/articles/account-verification) and credit card info are not required for Heroku, but unverified accounts have [limitations](https://devcenter.heroku.com/articles/limits). For example the number of apps (5 vs 100), custom domains, etc. You can still be on the free plan after verification, so it might make sense to do so. It is up to you.

You will need

git

to be [installed](https://github.com/git-guides/install-git) locally to be able to make changes to the git repository we are going to create.

To be able to communicate with Heroku from the command line, you'll use the Heroku Command Line Interface (CLI). Read more about [how to install it here](https://devcenter.heroku.com/articles/heroku-cli). Use

heroku --version

to test if the CLI is ready to be used. Then type

heroku login

which will prompt you to type in credentials. Next time the CLI will log you in automatically.

It goes without saying that you will need [R, shiny](https://hosting.analythium.io/the-anatomy-of-a-shiny-application/), and the [Docker Engine](https://hosting.analythium.io/docker-basics-for-data-apps/) installed too.

**The Shiny app**

Let's create a new folder called

heroku-hello

and inside a

Dockerfile

:

mkdir heroku-hello

cd heroku-hello

touch Dockerfile

We will use the [dockerized Hello example](https://hosting.analythium.io/dockerizing-shiny-applications/). This is an existing image that we use as a parent image in the

Dockerfile

:

FROM registry.gitlab.com/analythium/shinyproxy-hello/hello

ENV PORT=3838

CMD ["R", "-e", "shiny::runApp('/home/app', host = '0.0.0.0', port=as.numeric(Sys.getenv('PORT')))"]

The port settings and the

CMD

part is different from previous Dockerfiles that we used. The port number is passed as an environment variable by the [Heroku container runtime](https://devcenter.heroku.com/articles/container-registry-and-runtime#dockerfile-commands-and-runtime). Therefore the

runApp()

command needs to pick up the port number using

Sys.getenv('PORT')

. The reason we add

ENV PORT=3838

is to help with local testing.

If you want to apply this to your own app, you can

* use the dockerized version of your app as a parent image as we did above;
* or you can edit the Dockerfile to rely on the

$PORT

variable instead of hard coding and exposing a specific port.

Let's test the app:

docker build -t heroku-hello .

docker run -p 3838:3838 heroku-hello

Visit

127.0.0.1:3838

to see the familiar purple histogram with the sample size slider.

**Deployment**

Create a

heroku.yml

(

touch heroku.yml

) file in your application's root directory. The following example

heroku.yml

specifies the Dockerfile to be used to build the image for the app’s web process:

build:

docker:

web: Dockerfile

**Create the application**

This is the point where you have to commit your changes if any. Only repositories with new commits will deploy once the app is already on Heroku:

# Initialize the local directory as a Git repository

git init -b main

# Add the files and stage them for commit

git add .

# Commit tracked changes and sign-off the message

git commit -s -m "First commit"

Create the Heroku application with the container stack:

heroku create --stack=container

The command will give you the application URL:

Creating app... done, ⬢ morning-plateau-34336, stack is container

https://[morning-plateau-34336.herokuapp.com/](http://morning-plateau-34336.herokuapp.com/) | <https://git.heroku.com/morning-plateau-34336.git>

You might have noticed that a [Heroku git remote](https://devcenter.heroku.com/articles/git#creating-a-heroku-remote) is also added for the application to track changes.

You can configure an existing application to use the container stack using

heroku stack:set container

.

**Deploy the application**

Deploy your application to Heroku, replace

main

with your branch name if it is different:

git push heroku main

This will trigger a git push and a docker build on a remote Heroku server and docker push to the [Heroku container registry](https://devcenter.heroku.com/articles/container-registry-and-runtime):

Enumerating objects: 4, done.

Counting objects: 100% (4/4), done.

Delta compression using up to 4 threads

Compressing objects: 100% (3/3), done.

Writing objects: 100% (4/4), 1.09 KiB | 1.09 MiB/s, done.

Total 4 (delta 0), reused 0 (delta 0), pack-reused 0

remote: Compressing source files... done.

remote: Building source:

remote: === Fetching app code

remote:

remote: === Building web (Dockerfile)

remote: Sending build context to Docker daemon 3.072kB

remote: Step 1/3 : FROM registry.gitlab.com/analythium/shinyproxy-hello/hello

remote: latest: Pulling from analythium/shinyproxy-hello/hello

remote: 4363cc522034: Pulling fs layer

...

remote: Successfully built 8b891983b438

remote: Successfully tagged 6dc2e345582f143cd104cf98fec788c65e9a72a6:latest

remote:

remote: === Pushing web (Dockerfile)

remote: Tagged image "6dc2e345582f143cd104cf98fec788c65e9a72a6" as "[registry.heroku.com/morning-plateau-34336/web](http://registry.heroku.com/morning-plateau-34336/web)"

remote: Using default tag: latest

remote: The push refers to repository [registry.heroku.com/morning-plateau-34336/web]

remote: cec4817fd20b: Preparing

...

remote: Verifying deploy... done.

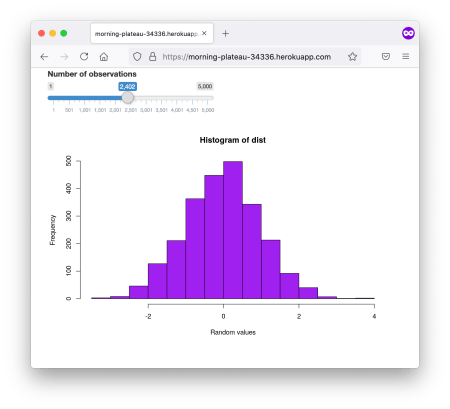
To https://[git.heroku.com/morning-plateau-34336.git](http://git.heroku.com/morning-plateau-34336.git)

\* [new branch] main -> main

Open the app in your browser using the

heroku open

command:



Log into your Heroku account and see the application listed in your dashboard. You can set up a [custom domain for the app](https://devcenter.heroku.com/articles/custom-domains) under the applications settings.

Note that there is no free SSL on custom domains, these will be served via HTTP and not HTTPS for apps that are using free resources.

If you want extra security, you have 3 options:

* upgrade to a paid Heroku tier;
* use [Cloudflare](https://support.cloudflare.com/hc/en-us/articles/205893698-Configure-Cloudflare-and-Heroku-over-HTTPS);
* serve the app via the original secure app URL and an iframe.

**Conclusions**

Heroku is really a no-hassle solution for hosting web applications, including dockerized Shiny apps. Using Docker makes it easy to rely on an already familiar workflow with very minimal modifications. The Heroku CLI is a powerful declarative tool that plays nicely with your git-based workflow.