Robust Data Pipelines with drake and Docker

@tudosgar | tamaszilagyi.com

The low-down

The end goal for most data analytics projects is to build a data product, whether it is a weekly dashboard or deploying a ML model.

From getting and cleaning the data to generating plots or fitting models, each project can be split up into individual tasks.

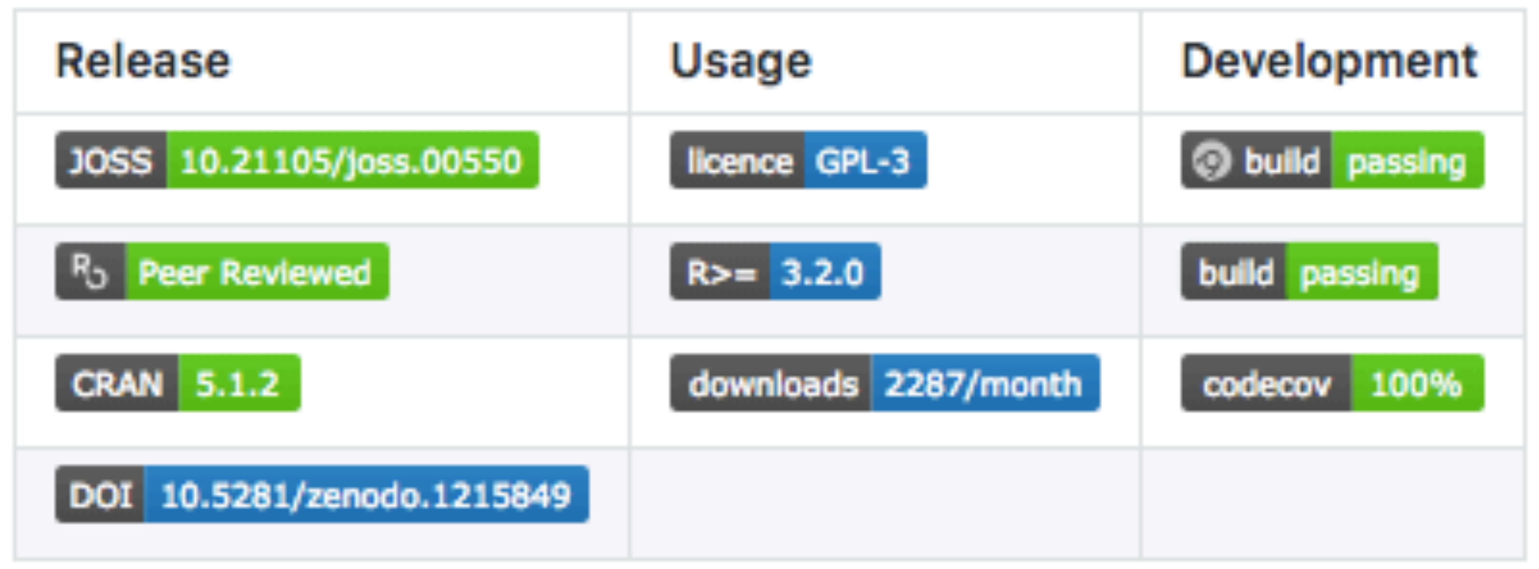
Afterwards, we want to make sure that the environment we wrote our code in can be replicated.

Automation.
Reproducibility.

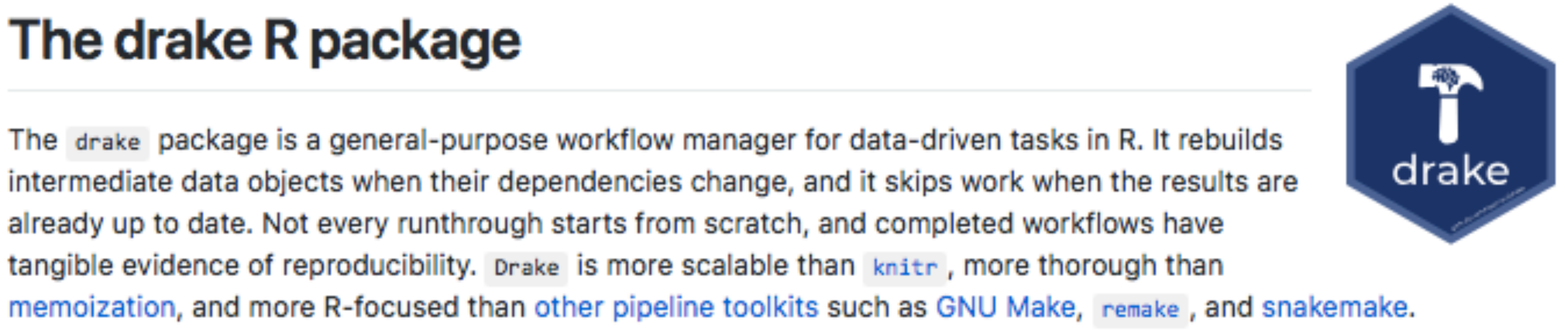
building pipelines

Workflow management tool that enables you to build managed workflows. It is a first serious attempt to create a tool like Airflow or Luigi, but in R

Scale up the work you need.
Skip the work you don't.
See evidence of reproducibility.



Release	Usage	Development
JOSS 10.21105/joss.00550	licence GPL-3	build passing
Peer Reviewed	R>= 3.2.0	build passing
CRAN 5.1.2	downloads 2287/month	codecov 100%
DOI 10.5281/zenodo.1215849		



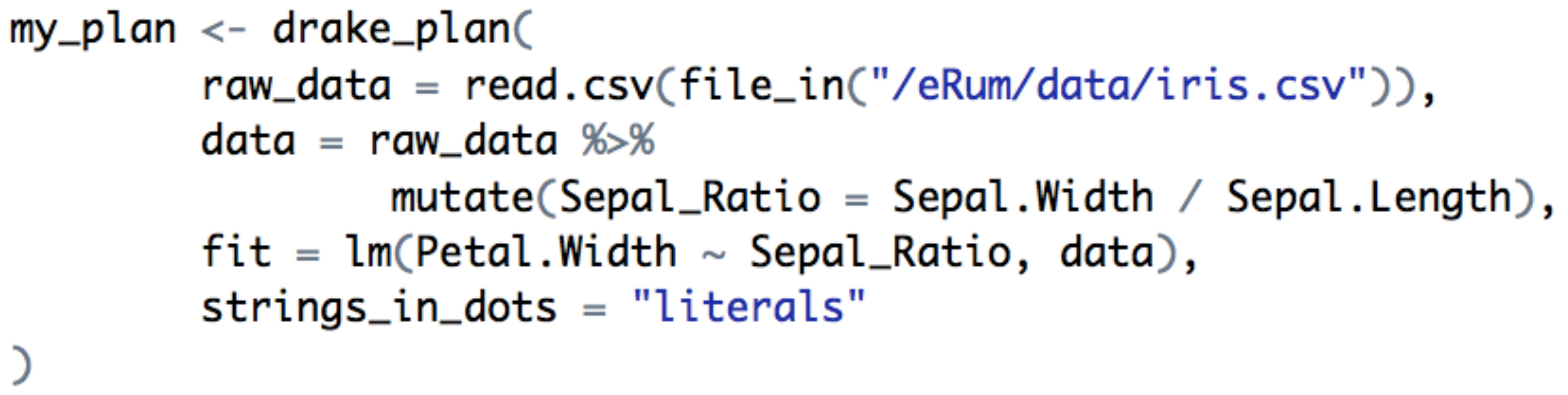
The drake R package

The `drake` package is a general-purpose workflow manager for data-driven tasks in R. It rebuilds intermediate data objects when their dependencies change, and it skips work when the results are already up to date. Not every runthrough starts from scratch, and completed workflows have tangible evidence of reproducibility. `Drake` is more scalable than `knitr`, more thorough than memoization, and more R-focused than other pipeline toolkits such as GNU Make, `remake`, and snakemake.

Your analysis is a sequence of transformations.

write a plan.

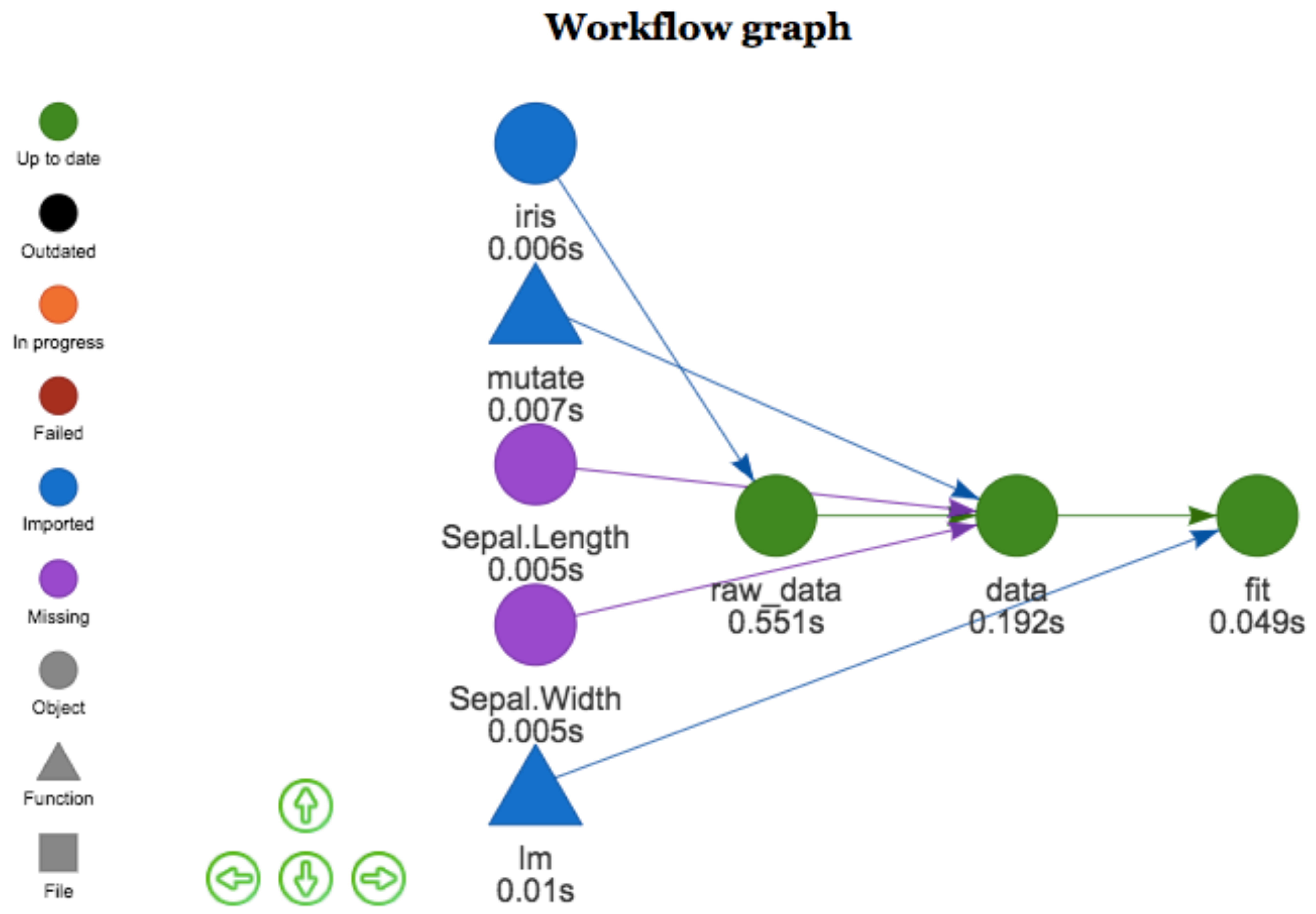
A plan can be simple, or very complex. It all depends on how you define the tasks' dependencies.



```
my_plan <- drake_plan(
        raw_data = read.csv(file_in("/eRum/data/iris.csv")),
        data = raw_data %>%
                mutate(Sepal_Ratio = Sepal.Width / Sepal.Length),
        fit = lm(Petal.Width ~ Sepal_Ratio, data),
        strings_in_dots = "literals"
)
```

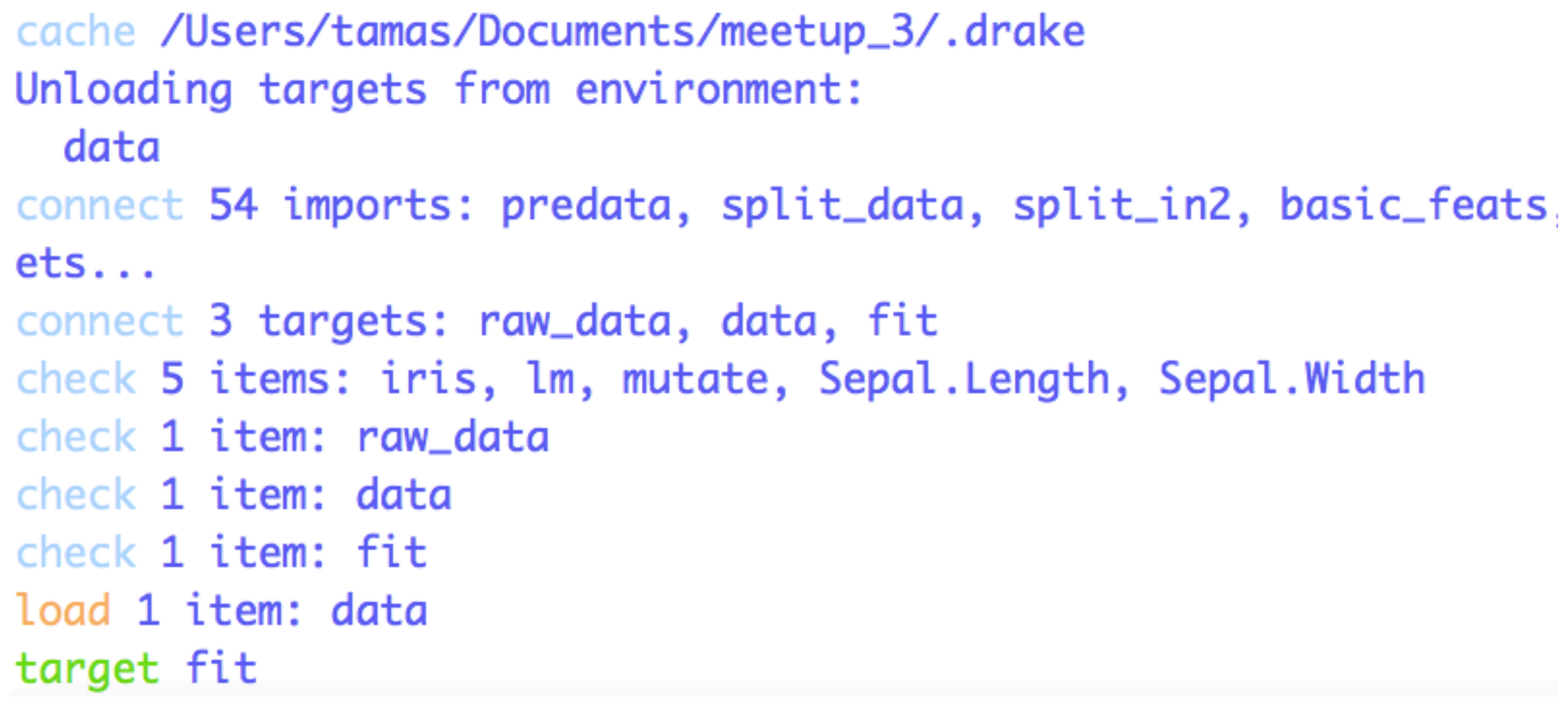
not only targets are kept track of.

By default we can visualise all imports, functions and transformations in our DAG, not only completed tasks. As a nice bonus we can also monitor progress for longer jobs.



make(my_plan)

Checks dependencies and cache before creating plan. This means that on subsequent runs, only the changed tasks will rerun, leaving the rest intact.



```
cache /Users/tamas/Documents/meetup_3/.drake
Unloading targets from environment:
  data
connect 54 imports: predata, split_data, split_in2, basic_feats
ets...
connect 3 targets: raw_data, data, fit
check 5 items: iris, lm, mutate, Sepal.Length, Sepal.Width
check 1 item: raw_data
check 1 item: data
check 1 item: fit
load 1 item: data
target fit
```

A container is kinda like a VM but...different.

benefits of containers

- Easily reproduce your infrastructure
- Runs independent of host OS
- Consistency in production

step 1:
Dockerfile

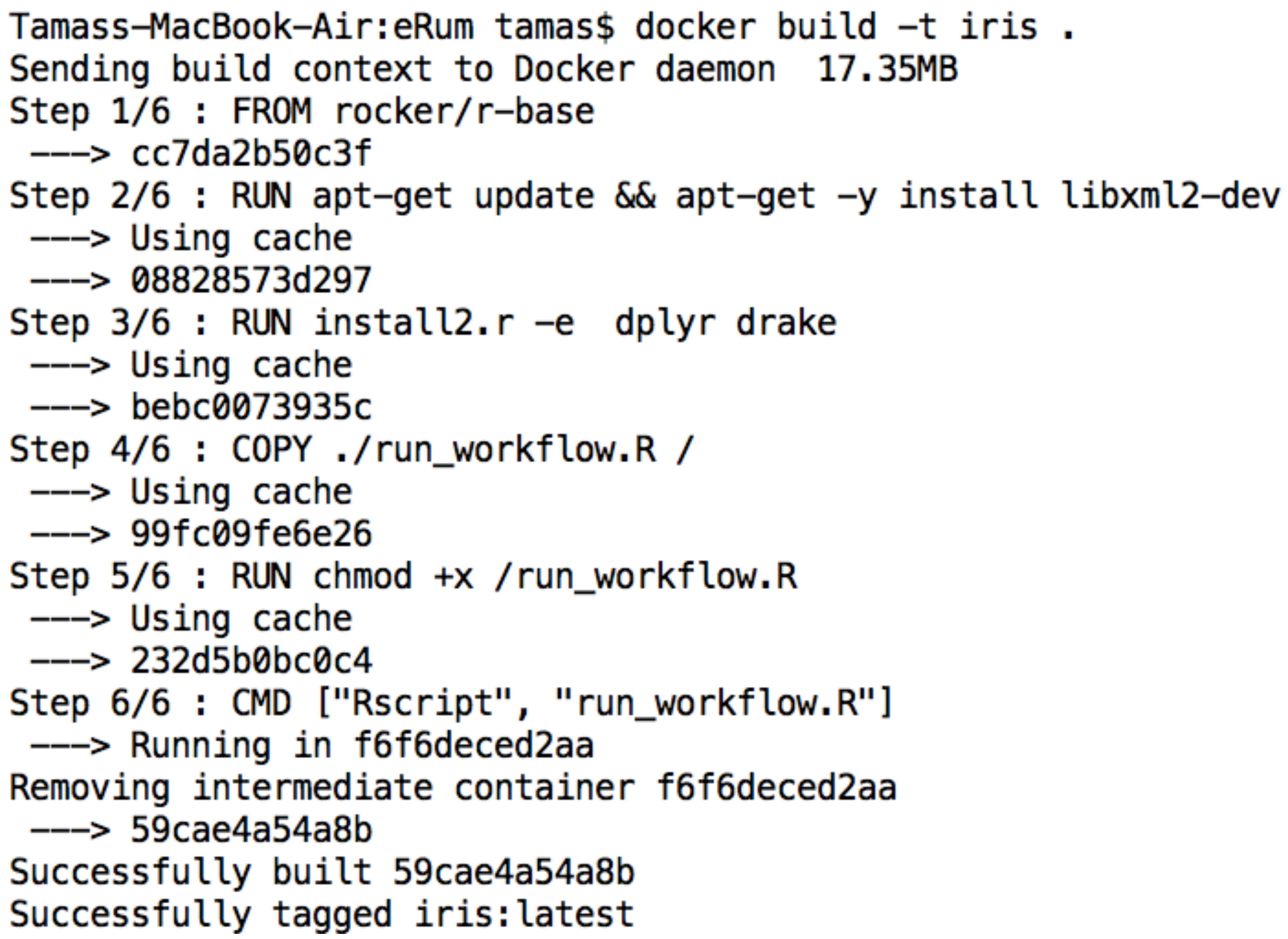
A Dockerfile includes, in backwards order:

- Your script
- Package dependencies of your script
- System level dependencies of these packages

```
FROM rocker/r-base

## install XML dependency
RUN apt-get update && apt-get -y install libxml2-dev
## install R packages from CRAN
RUN install2.r -e  dplyr drake

## Put script inside container
COPY ./run_workflow.R /
## Make scripts executabe
RUN chmod +x /run_workflow.R

## Run workflow
CMD ["Rscript", "run_workflow.R"]
```

step 2:
build image

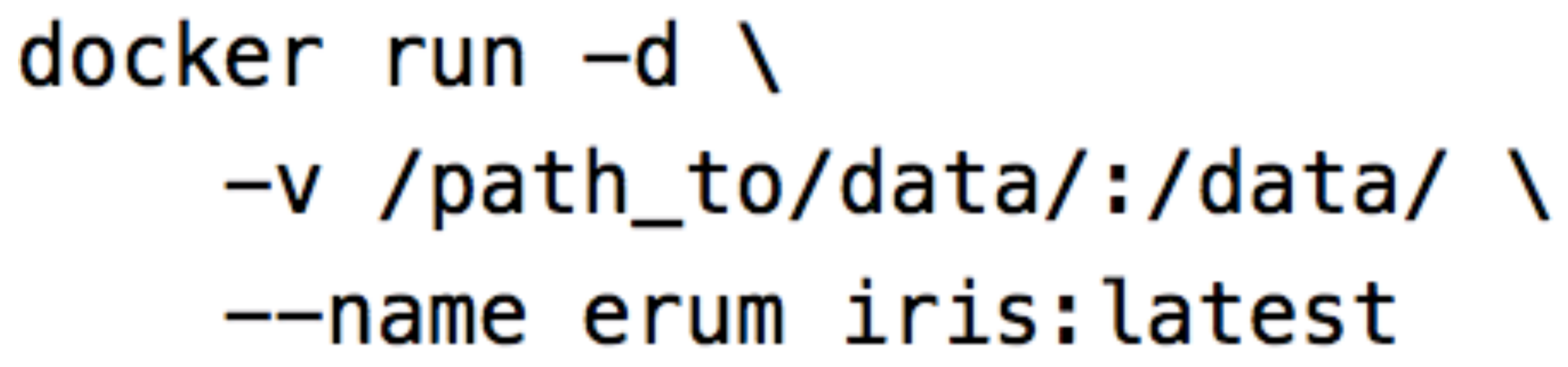
This is the stage where we actually build our *mini computer*, ready for deployment.



```
Tamass-MacBook-Air:eRum tamas$ docker build -t iris .
Sending build context to Docker daemon  17.35MB
Step 1/6 : FROM rocker/r-base
 ---> cc7da2b50c3f
Step 2/6 : RUN apt-get update && apt-get -y install libxml2-dev
 ---> Using cache
 ---> 08828573d297
Step 3/6 : RUN install2.r -e  dplyr drake
 ---> Using cache
 ---> bebc0073935c
Step 4/6 : COPY ./run_workflow.R /
 ---> Using cache
 ---> 99fc09fe6e26
Step 5/6 : RUN chmod +x /run_workflow.R
 ---> Using cache
 ---> 232d5b0bc0c4
Step 6/6 : CMD ["Rscript", "run_workflow.R"]
 ---> Running in f6f6deced2aa
Removing intermediate container f6f6deced2aa
 ---> 59cae4a54a8b
Successfully built 59cae4a54a8b
Successfully tagged iris:latest
```

step 3:
run container

Instantiates the container, mounts the folder from our host where the data resides and runs our executable as defined in the Dockerfile.



```
docker run -d \
    -v /path_to/data/:/data/ \
    --name erum iris:latest
```

Find me

http://tamaszilagyi.com/

@tudosgar

Thank you.

@tudosgar | tamaszilagyi.com