Setting up a package library

The RStudio project I provided already comes with a package library stored in the `renv.lock` file. If you prefer to work with your own app, you can create your own `renv.lock` file by installing the `renv` package from within your RStudio project and executing `renv::init()`. This initializes `renv` for your project and creates a `renv.lock` file in your project root folder. You can find more information on `renv` over at RStudio's introduction article on it.

The Dockerfile

The Dockerfile is once again the central piece of creating a Docker image. We now aim to repeat this process for an entire app where we previously only built a single script into an image. The step from a single script to a folder with multiple scripts is small, but there are some significant changes needed to make our app run smoothly.

```
# Base image https://hub.docker.com/u/rocker/
FROM rocker/shiny:latest

# system libraries of general use
## install debian packages
RUN apt-get update -qq && apt-get -y --no-install-recommends install \
    libxml2-dev \
    libcairo2-dev \
    libsqlite3-dev \
    libmariadbd-dev \
    libpq-dev \
    libssh2-1-dev \
    unixodbc-dev \
    libcurl4-openssl-dev \
    libssl-dev

## update system libraries
RUN apt-get update && \
    apt-get upgrade -y && \
    apt-get clean

# copy necessary files
## app folder
COPY /example-app ./app
## renv.lock file
COPY /example-app/renv.lock ./renv.lock

# install renv & restore packages
RUN Rscript -e 'install.packages("renv")'
RUN Rscript -e 'renv::restore()'

# expose port
EXPOSE 3838

# run app on container start
CMD ["R", "-e", "shiny::runApp('/app', host = '0.0.0.0', port = 3838)"]
```

The base image

The first difference is in the base image. Because we're dockerizing a ShinyApp here, we can save ourselves a lot of work by using the `rocker/shiny` base image. This image handles the necessary

dependencies for running a ShinyApp and comes with multiple R packages already pre-installed.

Necessary files

It is necessary to copy all relevant scripts and files for your app to your Docker image, so the Dockerfile does precisely that by copying the entire folder containing the app to the image.

We can also make use of `renv` to handle package installation for us. This is why we first copy the `renv.lock` file to the image separately. We also need to install the `renv` package separately by using the Dockerfile's ability to execute R-code by prefacing it with `RUN Rscript -e`. This package installation allows us to then call `renv` directly and restore our package library inside the image with `renv::restore()`. Now our entire project package library will be installed in our Docker image, with the exact same version and source of all the packages as in your local development environment. All this with just a few lines of code in our Dockerfile.

Starting the App at Runtime

At the very end of our Dockerfile, we tell the container to execute the following R-command:

```
shiny::runApp('/app', host = '0.0.0.0', port = 3838)
```

The first argument allows us to specify the file path to our scripts, which in our case is `./app`. For the exposed port, I have chosen 3838, as this is the default choice for RStudio Server, but can be freely changed to whatever suits you best.

With the final command in place every container based on this image will start the app in question automatically at runtime (and of course close it again once it's been terminated).

The Finishing Touches

With the Dockerfile set up we're now almost finished. All that remains is building the image and starting a container of said image.

Building the image

We open the terminal, navigate to the folder containing our new Dockerfile, and start the building process:

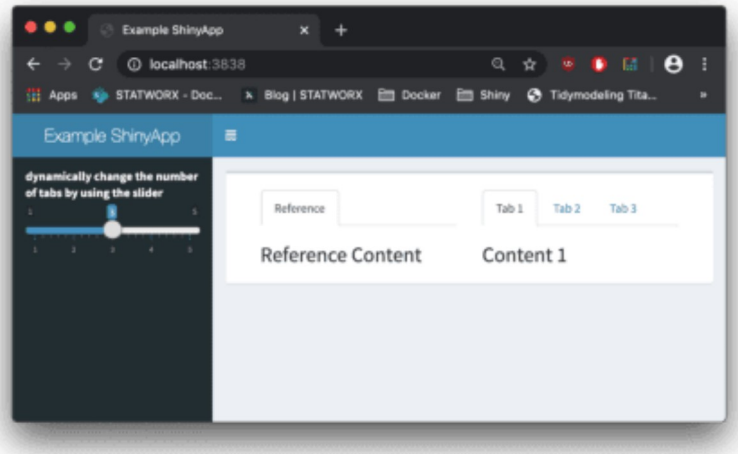
```
docker build -t my-shinyapp-image .
```

Starting a container

After the building process has finished, we can now test our newly built image by starting a container:

```
docker run -d --rm -p 3838:3838 my-shinyapp-image
```

And there it is, running on localhost:3838.



Outlook

Now that you have your ShinyApp running inside a Docker container, it is ready for deployment! Having containerized our app already makes this process a lot easier; there are further tools we can employ to ensure state of the art security, scalability, and seamless deployment. Stay tuned until next time, when we'll go deeper into the full range of RShiny and Docker capabilities by introducing ShinyProxy.