Shiny app deployment using ShinyProxy

Tobia De Koninck – Open Analytics

September 22, 2023

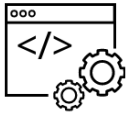
Open Analytics

Data Science Company

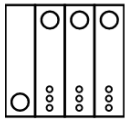
Data Science Company

STATISTICAL CONSULTING

SCIENTIFIC PROGRAMMING



APPLICATION DEVELOPMENT AND INTEGRATION



DATA ANALYSIS HARDWARE AND HOSTING

Agenda

Challenges when deploying Shiny apps in production:

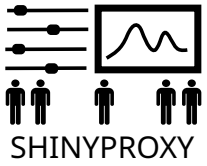
- containers
- installing R packages and system dependencies
- using apps with different R, Bioconductor or Shiny versions
- persistence & databases
- authentication & authorisation
- monitoring
- initialization times

Goal: give you inspiration and tips on what is possible with Shiny and ShinyProxy

ShinyProxy

- deploy data science apps in an enterprise context
- Shiny, Dash, Streamlit, Jupyter notebooks, IDEs, Gradio, Voila, Flask ...
- supports standard authentication/authorization technologies
- supports high availability
- scalable

https://shinyproxy.io



Scalability & use-cases

- single (Docker) server with one app and a few users
- public instance (cluster) with 50 apps and 300 users per day
- cluster with 200 users providing R and Python IDEs
- validated (GxP) environment with full audit logging

Containers

Containers

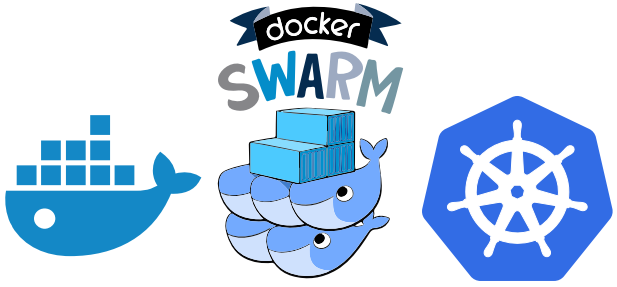
Docker

Docker makes it easy to create, deploy, and run applications using containers. It allows to combine an application with everything it needs, such as libraries, dependencies, configuration, ... into a container image.

The `Dockerfile` is a set of instructions telling Docker how to create the Docker image. For example, which R version you need, which R packages etc.

Why use Docker in ShinyProxy?

- the application is bundled and installed into a Docker image
- every time a user runs an application, a container spins up and serves the application
- fully isolated 'workspace' per session
- use different R, Shiny or Bioconductor versions on a single server, without installing this upfront on the server
- full control on memory and cpu usage
- archive every version of an application in order to achieve reproducibility
- Docker, Docker Swarm and Kubernetes supported



Extensible architecture: working on additional (cloud) container runtimes

Experience with Docker

Who is already using Docker? What do you like?

Experience with Docker

There is a learning curve, but in general we notice data scientists are able to quickly learn and use Docker in a good way.

In addition, because Docker makes it easier to reproduce results, it's very useful in (data) science.

Many data science departments are adopting containers.

Installing R packages and system dependencies

Working Example

In order to guide the discussion, we will use a working example:

interactiveDisplay

> Package for enabling powerful shiny web displays of Bioconductor objects

https://www.bioconductor.org/packages/release/bioc/html/interactiveDisplay.html

Create the Dockerfile

```
FROM rocker/r-ver:4.3.1

RUN apt-get update && \
    apt-get install -y --no-install-recommends libz-dev liblzma-dev libbz2-dev \
        libcurl4-openssl-dev libxt6

RUN R -q -e 'install.packages("BiocManager"); BiocManager::install(version = "3.17")'
RUN R -q -e 'BiocManager::install(c("GenomicRanges", "Gviz"), version = "3.17")'
RUN R -q -e 'BiocManager::install(c("ggbio", "interactiveDisplay"), version = "3.17")'

COPY Rprofile.site /usr/local/lib/R/etc/
EXPOSE 3838
CMD ["R", "-q", "-e", "library(interactiveDisplay);data(mmgr);display(mmgr);"]
```

System Dependencies

- find system dependencies of R packages on: https://packagemanager.posit.co/client/#/repos/2/packages/shiny
- overview of common system dependencies in R: https://r-universe.dev/sysdeps/
- all Ubuntu packages can be found on https://packages.ubuntu.com/
- https://rocker-project.org/use/extending.html

Often trial-and-error: both for CRAN and Bioconductor packages.

r2u

Fast, easy and good?

> Full integration with apt as every binary resolves all its dependencies: No more installations (of pre-built archives) only to discover that a shared library is missing. No more surprises.

https://eddelbuettel.github.io/r2u/

Very promising solution.

Supports Bioconductor

r2u

```
FROM rocker/r2u:jammy

RUN R -q -e 'install.packages("BiocManager"); BiocManager::install(version = "3.17")'
RUN R -q -e 'BiocManager::install(c("GenomicRanges", "Gviz"), version = "3.17")'
RUN R -q -e 'BiocManager::install(c("ggbio", "interactiveDisplay"), version = "3.17")'

COPY Rprofile.site /usr/lib/R/etc/
EXPOSE 3838
CMD ["R", "-q", "-e", "library(interactiveDisplay);data(mmgr);display(mmgr);"]
```

Easy: ✔
Fast: ✔ 1m36s vs 34m27s
Good: ✔

Pak

> pak installs R packages from CRAN, Bioconductor, GitHub, URLs, git repositories, local files and directories. It is an alternative to install.packages() and devtools::install_github(). pak is fast, safe and convenient.

https://pak.r-lib.org/

- Supports Bioconductor
- installs system dependencies as well: https://pak.r-lib.org/dev/reference/sysreqs.html
- automatically selects the Bioconductor version using the R version.
- ⚠️ pak did not install all required system dependencies -> trial and error still needed
- automatically parallelize package download, installation and compilation
- (slower than r2u: 13m14s)

Pak

```
FROM rocker/r-ver:4.3.1

RUN apt-get update && \
    apt-get install -y --no-install-recommends liblzma-dev libbz2-dev libxt6

RUN R -q -e 'install.packages("pak")'
RUN R -q -e 'pak::pkg_install(c("GenomicRanges", "Gviz", "ggbio", "interactiveDisplay"))'

COPY Rprofile.site /usr/local/lib/R/etc/
EXPOSE 3838
CMD ["R", "-q", "-e", "library(interactiveDisplay);data(mmgr);display(mmgr);"]
```

Renv

> The renv package helps you create reproducible environments for your R projects. Use renv to make your R projects more isolated, portable and reproducible.

> Reproducible: renv records the exact package versions you depend on, and ensures those exact versions are the ones that get installed wherever you go.

https://rstudio.github.io/renv/

- Supports Bioconductor
- can be combined with pak (although it is still very new and experimental)
- possible to select the Bioconductor version
- Dockerfile can be very similar for completely different projects

https://rstudio.github.io/renv/articles/bioconductor.html

Renv

```
FROM rocker/r-ver:4.3.1

RUN apt-get update && \
    apt-get install -y --no-install-recommends libz-dev libbz2-dev liblzma-dev \
        libcurl4-openssl-dev libxt6

RUN R -q -e 'install.packages("renv")'
RUN R -q -e 'renv:::renv_pak_init(stream = "devel", force = TRUE)'
RUN mkdir /app

COPY renv.lock /app
WORKDIR /app

RUN R -q -e 'renv::restore()'

COPY Rprofile.site /usr/lib/R/etc/
EXPOSE 3838
CMD ["R", "-q", "-e", "library(interactiveDisplay);data(mmgr);display(mmgr);"]
```

Installing packages and system dependencies

What tool do you use?

Which tool have you already tested?

Build the Dockerfile

```
docker build -t myregistry.com/interactive-display
docker push myregistry.com/interactive-display
```

Works on Linux, MacOS and Windows.

Build the Dockerfile

Building and pushing can and should be fully automated:

- store the `Dockerfile` in Git
- edit the file
- `git commit && git push`
- automatically start building the image

GitHub Actions

Jenkins

GitLab

ShinyProxy configuration

Update the ShinyProxy config

Edit the `application.yml` file and restart ShinyProxy.
As usual, best practice to automate this:

- store the `application.yml` in Git
- edit the file
- `git commit && git push`
- automatically deploy a new ShinyProxy server without downtime

```
specs:
  - id: interactiveDisplay
    display-name: InteractiveDisplay
    description: package for enabling powerful shiny web displays of Bioconductor objects
    container-image:  myregistry.com/interactive-display
```

Apps with different R, Bioconductor or Shiny versions

In essence this is solved by using Docker images

Let's deploy another app using an older R version:

```
FROM openanalytics/r-ver:3.6.3

RUN apt-get update && apt-get install -y --no-install-recommends \
    libz-dev

RUN R -q -e "install.packages('shiny')"

CMD ["R", "-q", "-e", "shiny::runExample('01_hello')"]
```

Reproducibility

- using Docker images already provides a lot of reproducibility: create once, run infinite times
- reproducing the Docker image itself: pin all versions in the `Dockerfile`
- allow users to run older versions of your app: GxP / clinical trials
- run multiple versions side-by-side
- related: when reproducibility is important, so are automatic tests

```
specs:
  - id: reproducibility
    max-instances: -1
    container-image: |
      "myregistry.com/img:#{proxy.getRuntimeObject('SHINYPROXY_PARAMETERS').getValue('version')}"
    parameters:
      definitions:
        - id: version
          default-value: v1.0.1
      value-sets:
        - values:
            version:
              - v1.0.1
              - v1.0.0
              - v0.9.0
```

Reproducibility

How do you guarantee reproducibility when using Shiny apps?
Do you write tests for your Shiny app?

Persistence & databases

Persistence for Applications

Many Shiny apps require a directory to store files.
ShinyProxy allows accessing a shared directory using volumes.

- shared between multiple users:

```
specs:
  - id: interactiveDisplay
    container-image: myregistry.com/interactive-display
    container-volumes: ["/mnt/interactive-display:/mnt/interactive-display"]
```

- unique to every user:

```
specs:
  - id: interactiveDisplay
    container-image: myregistry.com/interactive-display
    container-volumes: ["/mnt/interactive-display/#{proxy.userId}:/mnt/interactive-display"]
```

- essential when using ShinyProxy for hosting IDEs (Rstudio, Jupyter notebook ...)

Persistence for Applications

- use cloud-based filesystems, like AWS EFS or Azure FileShare
- supports PV & PVCs on Kubernetes
- often lacks file-locking support, e.g. SQLite -> use a network database
- file system permission can be tricky
- same mechanisms can be used for ephemeral storage

Do you face any other challenges with persistence storage?

Interaction with databases

- Shiny app can directly access any database service, using regular R code
- because Docker isolates the network, accessing databases on the same server as ShinyProxy can be tricky, but is possible
- best practice to pass database host, username and password as environment variables:

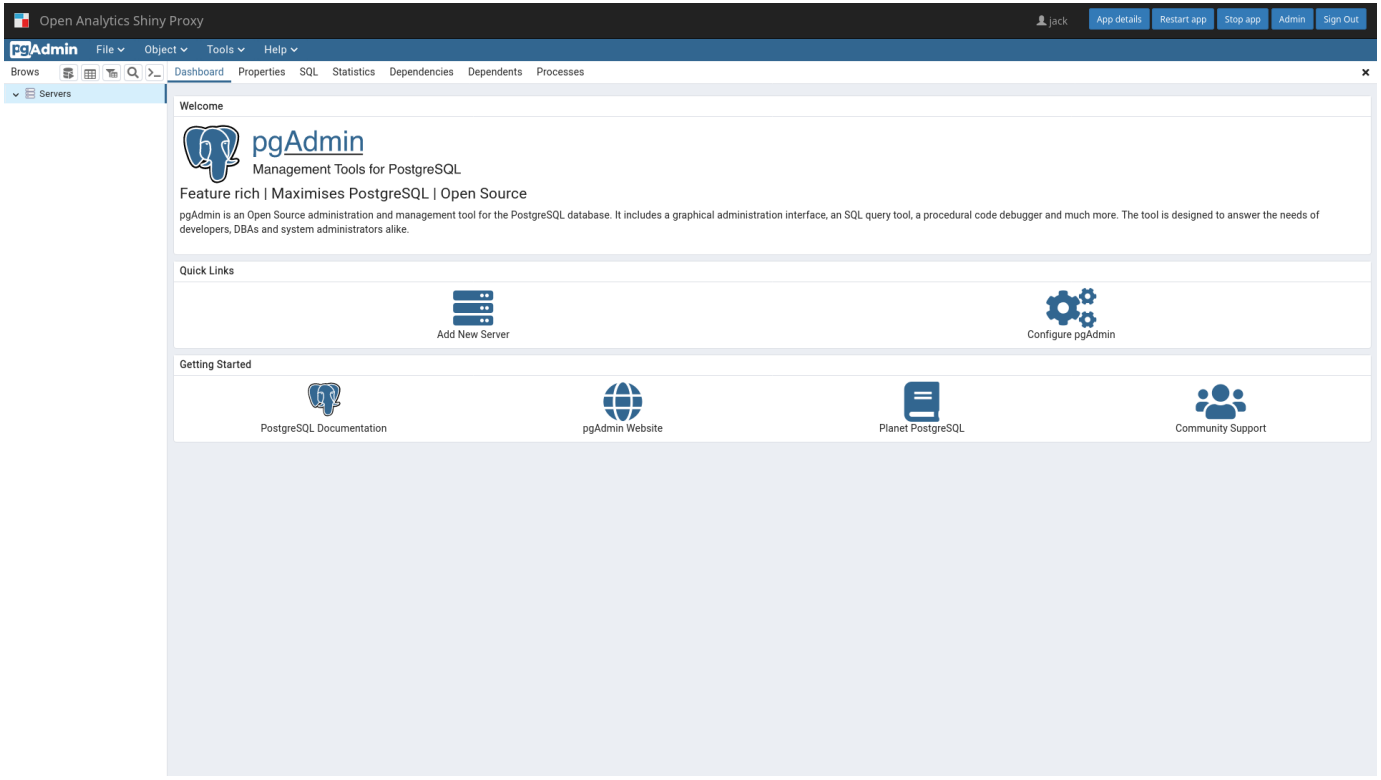
```
specs:
  - id: interactiveDisplay
    container-image: myregistry.com/interactive-display
    container-env:
      DB_HOST: "172.16.10.10"
      DB_USERNAME: "postgres"
      DB_PASSWORD: "myPassword!"
```

- Kubernetes secrets natively supported
- Docker swarm secrets natively supported
- ⚠️ do not store the password in the R code or on git

Interaction with databases

Bonus: run pgadmin on ShinyProxy:

https://github.com/openanalytics/shinyproxy-pgadmin-demo

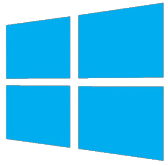


Authentication & Authorisation

Authentication

> Proving a user's identity

- best-practice: authentication should not happen in the Shiny app
- don't reinvent the wheel: OpenID Connect, LDAP, SAML
- integrating with external tools allows for e-mail verification, resetting passwords, 2FA ...
- ShinyProxy authenticates the user and forwards this information to the container using environment variables



Active Directory

Authentication

```
library(shiny)

shinyServer(function(input, output, session) {

  output$username <- renderText({ Sys.getenv("SHINYPROXY_USERNAME") })
  output$groups <- renderText({ Sys.getenv("SHINYPROXY_USERGROUPS") })

})
```

Authorisation

> Giving a user permission to access or do something

ShinyProxy: whether a user can access an app:

```
specs:
  - id: auth-demo
    display-name: Demo of authentication and authorisation
    container-image: myregistry.com/auth-demo
    access-users:
      - jack
    access-groups:
      - mathematicians
```

Authorisation

Shiny: whether a user can see some data or preform a certain action:

Monitoring

Logs & Metrics

https://shinyproxy.io/documentation/usage-statistics/

https://github.com/openanalytics/shinyproxy-monitoring

Usage statistics, logging and audit logging supported.

Initialization time

Problem

- a new container is created when the user opens an app
- the container is created for a specific user –> can be customized for this user
- user has to wait for container and the R process to start

Solution

Solution: container pre-initialization

- ShinyProxy maintains a pool of containers, which are ready to use
- automatically scale-up and scale-down this pool
- automatically re-create containers when the configuration changes
- will support multiple users using a single container

Thanks

- https://shinyproxy.io/
- https://shinyproxy.io/documentation/demos/
- Feature requests or bug reports: https://github.com/openanalytics/shinyproxy
- Support: https://support.openanalytics.eu/
- https://openanalytics.eu/
- info@openanalytics.eu