



Déployer une app shiny !!!

20-06-2022

Officiellement : Bonjour !

**

A propos de ThinkR

ThinkR est une société composée d'experts, leaders de la formation R en France.

ThinkR réalise également :

- des prestations de développement R,
- des accompagnements aux bonnes pratiques de développement.



A propos de moi

- D'où je viens ?
- Ce que j'ai fait avant d'être devant vous ?
- Depuis combien de temps je suis chez ThinkR ?
- Pourquoi aujourd'hui je vous forme à R ?

Timing

Modalités pratiques

Concentration & Pauses

Pause de 5 minutes toutes les 60 minutes.

<https://thinkr-open.github.io/timer>

- Pourquoi une pause de 5 minutes ?

Temps nécessaire pour "décrocher" (à condition de vraiment décrocher de son écran) tout en se souvenant ce qu'on était en train de faire avant de partir en pause.



- Pourquoi toutes les 60 minutes ?

Temps maximum d'attention.



N'hésitez pas à le rappeler à votre formateur·ice s'il oublie et que vous en ressentez le besoin.

Si vous avez besoin d'une pause avant, dites-le lui aussi !

Workshop User2022

Comment déployer une application shiny



LES OBJECTIFS DE CE WORKSHOP

- Comprendre et mettre en oeuvre un déploiement d'une application {shiny}
- Comprendre l'intérêt de {golem} pour déployer une applivation
- Savoir déployer une application sur Connect
- Dockeriser une application {shiny}
- Savoir déployer une application avec ShinyProxy

Les outils



- Une application {shiny} développée avec {golem}: dossier `hangman`
- Un Rstudio pour les manipulations <https://rstudio.cervangirard.me>
- Un connect pour le déploiement <https://connect.cervangirard.me>
- Un ShinyProxy pour le déploiement
 - <https://sp.cervangirard.me/sp-votrenom>
 - exemple pour cervan : <https://sp.cervangirard.me/sp-cervan>

Réglardons l'application à déployer



- Ouvrir le projet rstudio du dossier hangman
- Lancer l'application avec `golem::run_dev.R`



Une app avec {golem}, c'est quoi ?

Introduction à {golem}

Un outil pour les gouverner tous!



{golem} est un package R destiné à la création d'applications Shiny pour la production.

Utiliser ce package impose un cadre relativement strict, mais permet de se passer des réflexions techniques d'infrastructure.

Même si le framework peut sembler imposant de prime abord, en pratique le workflow est assez simple à suivre.

Installer {golem}

```
# install.packages("remotes")
# remotes::install_github("Thinkr-open/golem") # version de dev
install.packages("golem")
```

Pourquoi un package ?



Développer une application Shiny dans un package permet de :

- Documenter son code.
- Utiliser les infrastructures de test natives.
- Faciliter la maintenance sur le long terme.
- Gérer ses fichiers de manière efficace.
- Bénéficier des outils "classiques" de développement (`{devtools}`, `{usethis}`, `{testthat}`, ...).
- Standardiser le déploiement sur les différentes plateformes (RStudio Connect, ShinyApps.io, via Docker...)

Lancer {golem}



Structure d'un golem



DESCRIPTION

```
| -- dev/  
|   |-- 01_start.R  
|   |-- 02_dev.R  
|   |-- 03_deploy.R  
|   |-- run_dev.R  
|-- inst/  
|   |-- app  
|     |-- www/  
|       |-- favicon.ico  
|-- man/  
|   |-- run_app.Rd
```

NAMESPACE

```
monapp.Rproj  
| -- R/  
|   |-- app_config.R  
|   |-- app_server.R  
|   |-- app_ui.R  
|   |-- run_app.R
```

- DESCRIPTION & NAMESPACE : Méta données du package.
- dev/ : workflow de développement.
- inst : ressources externes.
- man : documentation de l'app, se génère automatiquement.
- monapp.Rproj : projet RStudio.
- R/app_config.R: fichier interne de configuration.
- R/app_server.R, app_ui.R : Fichiers contenant des fonctions pour construire server et ui.
- R/run_app.R : fonction qui va configurer et lancer l'app.



Commençons par remplir les métadonnées (une fois pour toute) de notre package :

```
golem::fill_desc(  
  pkg_name = "nasapp",  
  pkg_title = "Visualisation en Direct de l'ISS",  
  pkg_description = "Visualiser en Direct la position de l'ISS",  
  author_first_name = "colin",  
  author_last_name = "fay",  
  author_email = "colin@thinkr.fr",  
  repo_url = NULL  
)
```



```
usethis::use_mit_license(name = "Colin Fay")
usethis::use_readme_rmd()
usethis::use_code_of_conduct()
usethis::use_lifecycle_badge("Experimental")
usethis::use_news_md()

usethis::use_data_raw()

golem::use_recommended_tests()

golem::use_recommended_deps()

golem::use_utils_ui()
golem::use_utils_server()

golem::use_favicon( path = "path/to/favicon" )
```



Engineering de l'application, au quotidien :

```
usethis::use_package( "thinkr" )

golem::add_module( name = "name_of_module1" )

golem::add_fct( "helpers" )
golem::add_utils( "helpers" )

golem::add_js_file( "script" )
golem::add_js_handler( "handlers" )
golem::add_css_file( "custom" )

usethis::use_data_raw( name = "my_dataset", open = FALSE )
```



À la fin, une fois que l'on souhaite déployer :

```
devtools::check()  
rhub::check_for_cran()  
  
golem::add_rstudioconnect_file()  
golem::add_shinyappsi_file()  
golem::add_shinyserver_file()  
  
golem::add_dockerfile()  
golem::add_dockerfile_shinyproxy()  
golem::add_dockerfile_heroku()
```

Et maintenant ?



Comment compléter votre `{golem}` ?

- `app_ui.R` recevra la partie UI
- `app_server.R` recevra la contrepartie serveur

RStudio Connect

Diffuser ses applications



Le but de Rstudio Connect :

- Partager votre travail sans effort
- Faciliter la gestion des dépendances
- Gagner du temps dans le déploiement



Permet de déployer des applications shiny mais pas seulement.

- Les rmarkdowns sont très bien intégrés avec beaucoup de fonctionnalité
- Possibilité de déployer des applications python

Regardons à quoi cela ressemble :

<https://connect.cervangirard.me>

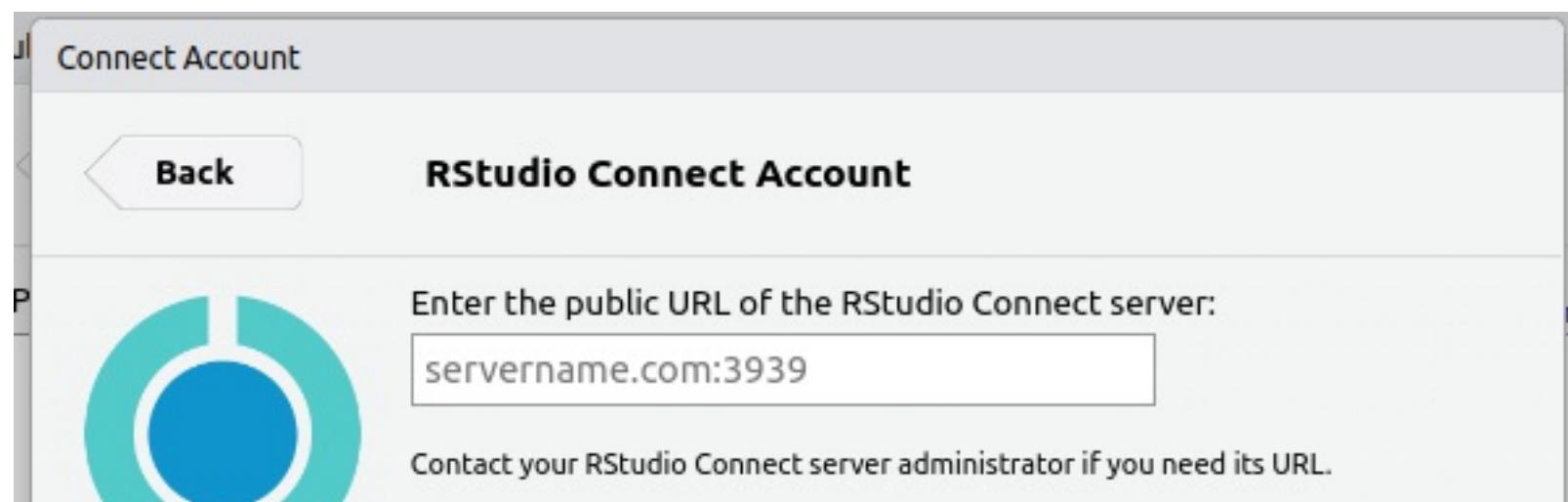
Dans un premier temps, regarder votre formateur faire et puis on vous laissera le temps de le refaire :) .

Déployons un Rmd ensemble :

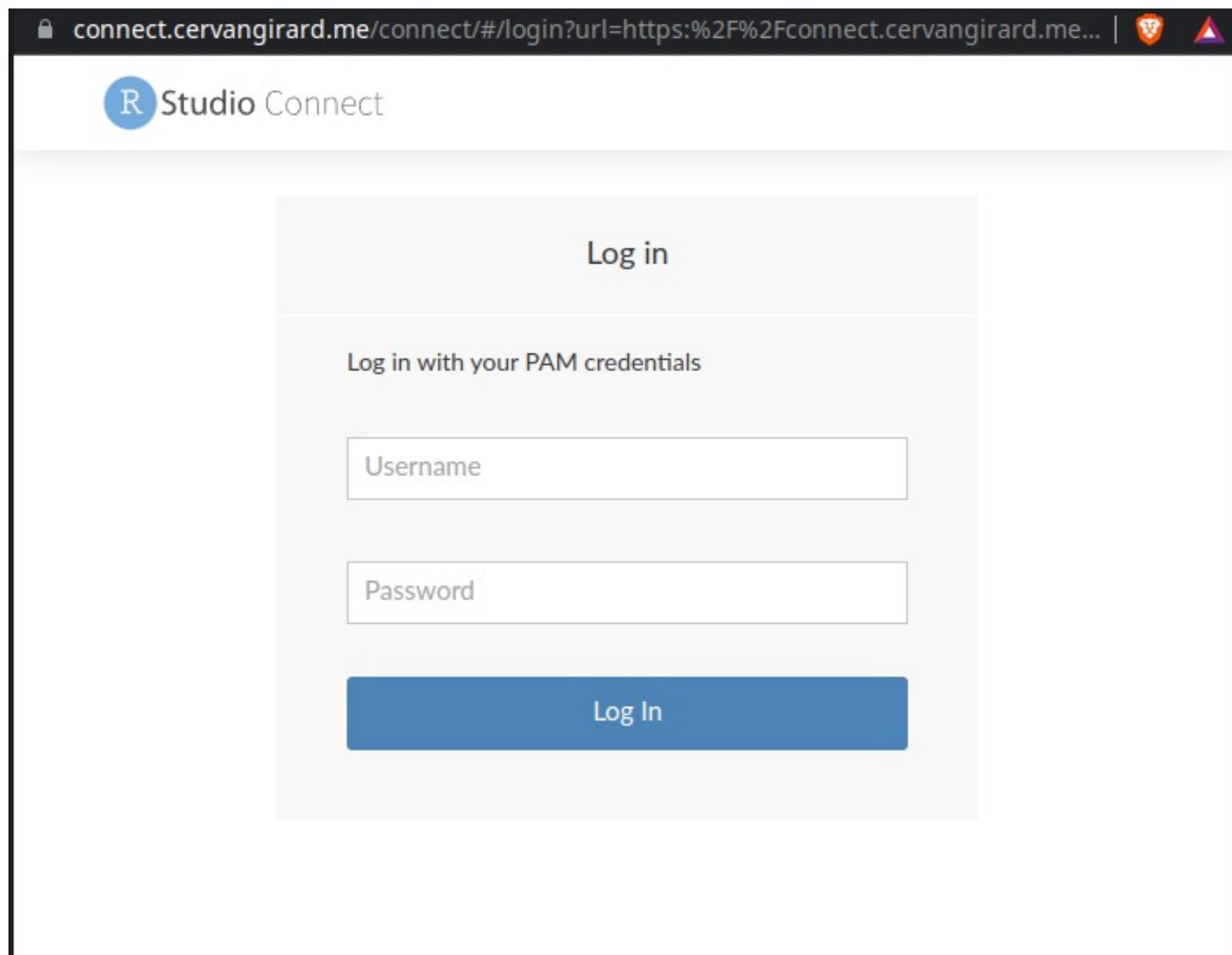
- Créer un nouveau Rmd
 - file > New file > Rmarkdown
- Cliquer sur le bouton bleu
- Choisir "Publish just this document"

Il faut maintenant ajouter notre Connect :

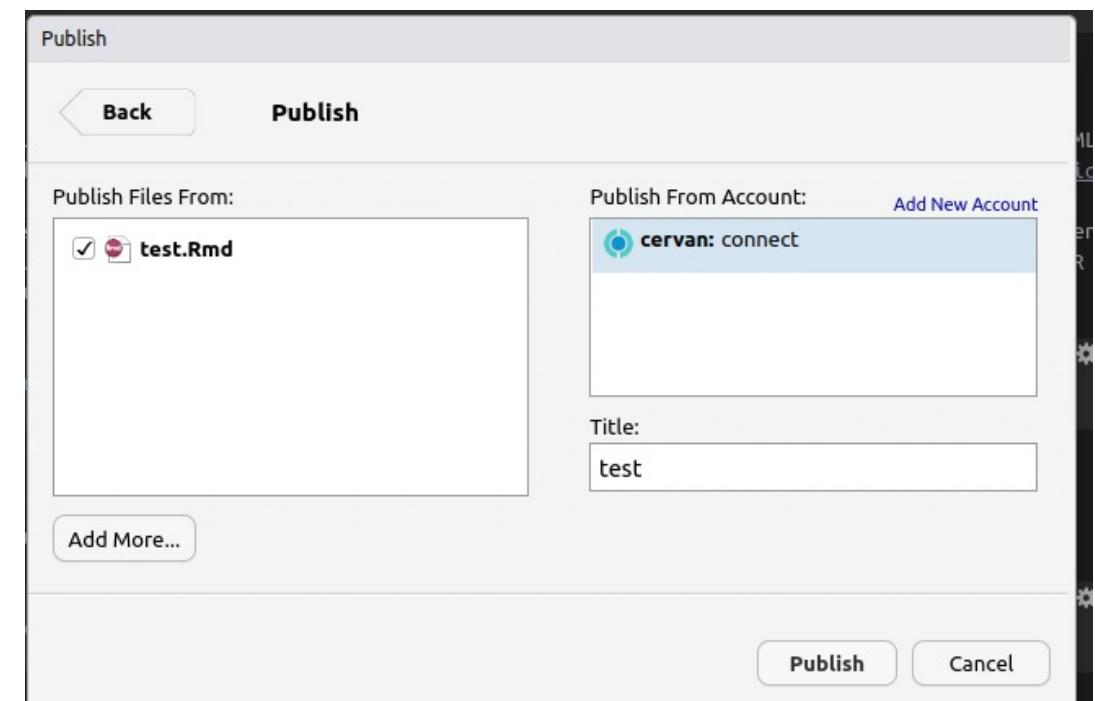
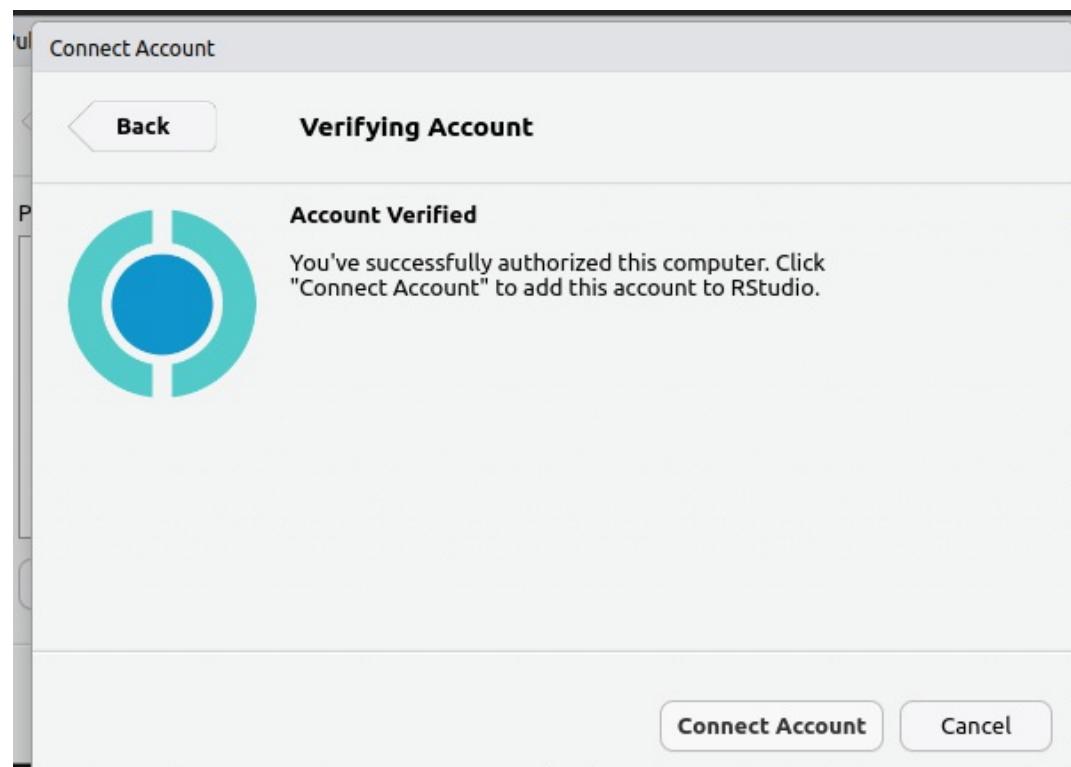
- Ajouter l'adresse suivante `connect.cervangirard.me` ici :



- Se connecter à Connect avec votre identifiant et votre mot de passe



- Quand vous avez réussi à vous connecter, vous pouvez alors choisir votre Connect



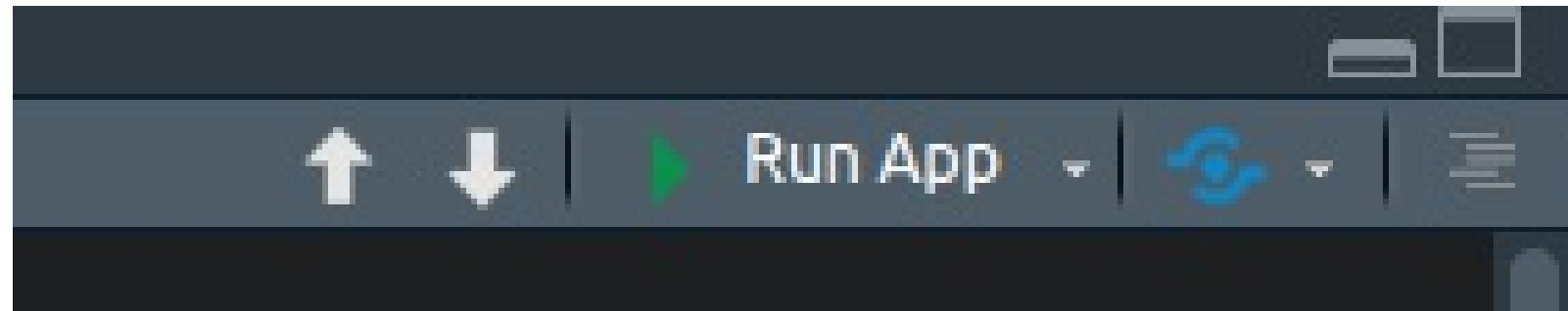
- Cliquer sur "Publish"

Comment déployer une application {shiny} avec {golem} et RSconnect



Rstudio intègre déjà des fonctionnalités pour déployer avec Connect.

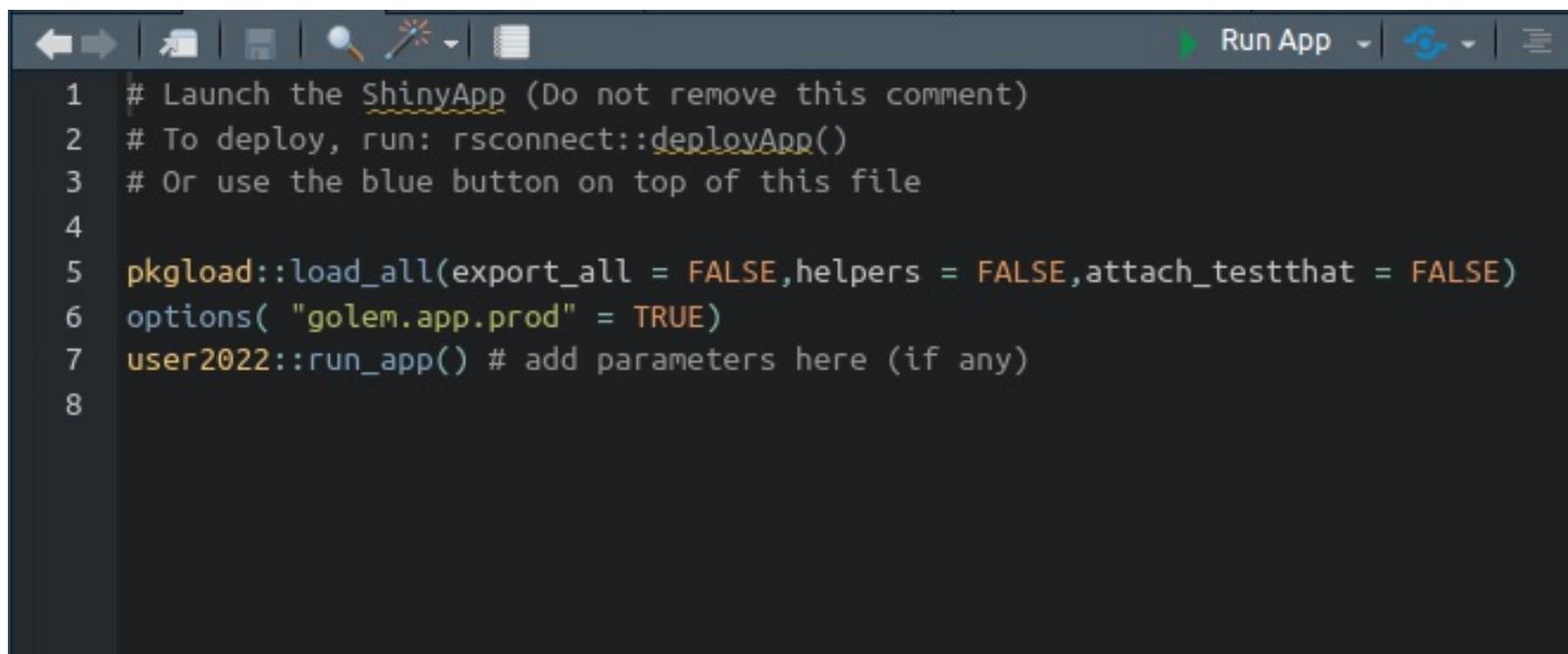
Il va reconnaître que vous manipuler une application et il va alors vous proposer "le bouton bleu" !



De même pour {golem}, des fonctionnalités permettent d'ajouter les fichiers nécessaires au déploiement.

Dans un premier temps, regarder votre formateur faire et puis on vous laissera le temps de le refaire :).

- Ajouter le fichier `app.R` avec `golem::add_rstudioconnect_file()`
- Ouvrir le fichier `app.R`



```
1 # Launch the ShinyApp (Do not remove this comment)
2 # To deploy, run: rsconnect::deployApp()
3 # Or use the blue button on top of this file
4
5 pkgload::load_all(export_all = FALSE, helpers = FALSE, attach_testthat = FALSE)
6 options("golem.app.prod" = TRUE)
7 user2022::run_app() # add parameters here (if any)
8
```

- Cliquer sur le bouton bleu.

Et sans bouton bleu magique ??



Le bouton magique est intégré à Rstudio mais comment faire sans lui ?

Le package `{rsconnect}` est fait pour ça !

Déployer une application avec

{rsconnect}



Dans un premier temps, regarder votre formateur faire et puis on vous laissera le temps de le refaire :).

- Aller sur connect
- Creer un token
- Se connecter au serveur avec {rsconnect}

```
rsconnect::addServer("https://connect.cervangirard.me/_api_", name = "connect")
rsconnect::connectApiUser(account = "cervan", server = "connect", apiKey =
Sys.getenv("API_CONNECT"))
```

- Déployer son application

```
rsconnect::deployApp(
  account = "cervan",
  server  = "connect"
)
```

BONUS : Déployer notre application



avec un CI

Voici les étapes:

- Ajouter un script dans le dossier `dev /deploy.R` avec les commandes précédentes
- Ajouter un `gitlab-ci.yml` et l'amender avec une étape `deploy`
- Faire en sorte que cette étape lance notre script `dev/deploy.R`

Shinyproxy et docker !Une autre solution pour déployer

Une autre solution pour déployer

Shinyproxy et docker !

Pour pouvoir déployer une application sur ShinyProxy, il faut connaitre docker !

Disclaimer: le but de ce tuto n'est pas de vous former à docker, il faudra approfondir de votre côté mais nous allons tenter de vous donner les bases pour déployer une app avec ShinyProxy

Docker

Moby dock, votre nouvelle meilleure amie

Docker est un logiciel libre relativement récent (2013), plébiscité par sa "simplicité" et son efficacité.



" Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur.".

Wikipedia

Docker en quelques mots :

- Permet d'empaqueter et de lancer un ou plusieurs applicatifs
- Fonctionne avec différents système d'opération (OS)
- Peut être lancé en local, ou sur un serveur
- Partage de manière intelligente les resources de la machine

Pourquoi utiliser Docker (avec R) ? :

- simplifie/supprime les soucis de dépendances
 - isole complètement un environnement
 - facilite la portabilité de l'application
 - la solution est réutilisable et amendable
- > permet de déployer votre application R (shiny ou non) partout où docker peut être installé (c'est-à-dire à peu près partout).
- > beaucoup plus léger et rapide qu'une machine virtuelle.

Docker, exemples

- ShinyProxy
- r-online
- r-ci
- r-db

Docker, terminologie

Docker repose sur le principe de conteneurs.

Un conteneur va "contenir" une application et toutes les dépendances nécessaires à son bon fonctionnement.

Notions importantes :

- conteneurs : ce qui va faire tourner les applications.
- images : environnement pré-conçues qui décrivent ce que devront faire les conteneurs.
- registry : dépôt qui regroupe les images de la communauté
- dockerfile : fichier texte qui correspond à la « recette » d'une image (permet de construire une image).

Principales commandes

```
docker pull <utilisateur>/<nom_image>
```

-> Télécharge l'image mis à disposition par l' sur le registry.

Exemple : `docker pull thinkr/rfull`

-> télécharge une image dédié à R proposée par thinkr.

```
docker pull alpine.
```

-> une image très légère de linux, pas besoin de préciser l'utilisateur, car c'est une image « officielle »

L'instruction pull ne fait que télécharger une image, aucun conteneur n'est fabriqué. Pour fabriquer (télécharger et lancer) un conteneur on peut utiliser directement docker run

```
docker run -it ubuntu
```

La dernière ubuntu est téléchargée, et lancée en donnant la main à l'utilisateur.

Les principaux arguments de docker run

- -i : lance le conteneurs en mode interactif.
- -t : à utiliser avec -i pour avoir un terminal (it).
- -p : connecte un port de la machine sur un port du conteneurs.
- -d : lance le conteneurs en mode « détaché »
- -v : monte un dossier du conteneurs avec un dossier de la machine.
- --name : donne un nom au conteneurs.

Exemple : Que fait l'instruction suivante ? :

```
docker run -d -e PASSWORD=coucou -p 80:8787 rocker/tidyverse
```

Gérer les conteneurs

```
docker ps
```

-> liste les conteneurs qui tournent sur la machine.

```
docker ps -a
```

-> liste tout les conteneurss, même ceux qui ne tournent pas.

```
docker stop <conteneurs-id>
```

-> stop le conteneurs qui était en train de tourner (docker kill en est la version brutale)

```
docker start <conteneurs-id>
```

-> relance le conteneurs qui était en train de tourner.

```
docker exec -it <conteneurs-id> bash
```

-> ouvre un terminal dans un conteneurs en cours d'exécution.

Docker

```
docker rm <conteneurs-id>
```

-> efface un conteneur.

```
docker rmi <nom_image>
```

-> efface une image.

```
docker rm -f $(docker ps a q)
```

-> efface TOUS les conteneurss.

```
docker rmi -f $(docker images q)
```

-> efface TOUTES les images.

Et R dans tout ca ?

Il est très ais  de construire sa propre image docker dont le but sera d'afficher votre application shiny.

Le plus simple est de partir depuis une image pr existante et de la modifier pour qu'elle convienne   notre besoin.

<https://hub.docker.com/u/rocker/>

Le projet rocker regroupe un certain nombre d'images pr -con ues qui vous permettent d'utiliser R dans docker (sans devoir tout monter depuis un linux « vide »).

Créer sa propre image

-> DOCKERFILE.

Construire sa propre image docker se résume à écrire un fichier DOCKERFILE, ce fichier contient la liste d'instruction à exécuter dans un conteneurs existant pour y installer tout ce dont vous avez besoin pour lancer votre application.

Un DOCKERFILE est une succession d'instruction, 1 ligne = 1 instruction. Chaque instruction commence par une de ces balises :

- FROM : précise l'image de base qui sera utilisée en point de départ.
- MAINTAINER : précise l'auteur de l'image.
- RUN : lance une instruction bash classique.
- COPY : copie un fichier de la machine vers l'image.
- EXPOSE : ouvre des ports sur la machine.
- CMD : indique la commande à exécuter juste après avoir lancé un conteneurs.

Dockerfile

Dans un fichier DOCKERFILE.

```
FROM rocker/r-ver  
CMD ["R", "-e", "message('Coucou depuis docker')"]
```

-> Après avoir trouver sa recette de cuisine, on peut construire notre image qui sera lancé par notre conteneur avec `docker build`.

Dans le terminal, en se mettant dans le même dossier que le fichier DOCKERFILE :

```
docker build -t coucou .  
docker run coucou
```

Résumons

- On déclare notre recette de cuisine dans un Dockerfile
- On construit une image avec ce Dockerfile
- On peut lancer un conteneur qui exécutera cette image

Créer sa propre image avec {dockerfiler}

```
library(dockerfiler)
```

```
mine <- Dockerfile$new()  
mine$RUN("apt-get install libxml2 -y")  
mine$RUN(r(install.packages("shiny")))
```

```
mine
```

```
#> FROM rocker/r-base  
#> RUN apt-get install libxml2 -y  
#> RUN R -e 'install.packages("shiny")'
```

- Faire un Dockerfile qui imprime un message depuis R : "Coucou votre prénom"
- Construire l'image qui devra s'appeler `votreprenom_coucou`
- Lancer cette image dans un conteneur

Encapsuler une application shiny {golem} avec docker

**

Encapsuler une application shiny

Pour diffuser une application shiny, il « suffit » de l'intégrer à un package R avec `{golem}`. Il est alors possible de bénéficier de tous les avantages des packages :

- Gestion des dépendances.
- Gestion des versions.
- Facilité d'installation et déploiement.
- gestion de la documentation

{golem} et docker

Une application shiny développée avec {golem} est avant tout un package et vous avez à votre disposition une fonction `run_app` qui lance l'application.

Grâce à {golem}, vous avez aussi des outils pour vous faciliter la vie pendant votre déploiement.

Construire une Dockerfile pour un {golem} revient à faire :

- `golem::add_dockerfile()`

Dans un premier temps, regarder votre formateur faire et puis on vous laissera le temps de le refaire :).

- Se mettre dans votre dossier `hangman`
- Construire la dockerfile de l'application `hangman` avec

```
golem::add_dockerfile(port = "choisir un port entre 32000 et 33000")
```

- Construire son image avec `docker build`, le nom doit être `votreprenom_hangman`
- Lancer L'image avec `docker run` et aller sur `sp.cervangirard.me:le port choisi`

Et ShinyProxy dans tout ça ?

**

Shinyproxy et docker !

Nous venons de faire le plus dure pour déployer une application Shiny avec ShinyProxy:

Etre capable de lancer son application avec docker !

Shiny Proxy

Shinyproxy



The header of the ShinyProxy dashboard. It features a blue bar at the top with the ShinyProxy logo (a graduation cap icon) and the word "SHINYPROXY". To the right is a search bar with a magnifying glass icon and the placeholder text "Search".

ShinyProxy

- [About](#)
- [Getting Started](#)
- [Deploying Apps](#)
- [Configuration](#)
- [ShinyProxy in a Container](#)
- [Security](#)
- [Usage Statistics](#)
- [Downloads](#)
- [Troubleshooting](#)
- [Support](#)

About

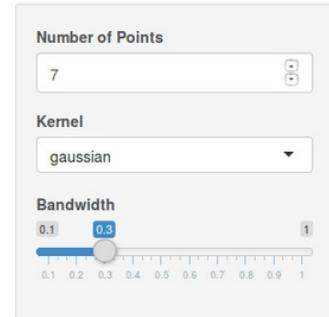


Density Estimation Explained

Number of Points
7

Kernel
gaussian

Bandwidth
0.1 0.3 1



A form interface for density estimation. It includes a dropdown for "Number of Points" set to 7, a dropdown for "Kernel" set to "gaussian", and a slider for "Bandwidth" set to 0.3. Below the form is a plot showing a black bell-shaped curve representing the density estimate. Three points are plotted on the x-axis: a red dot at approximately 0.3, a grey dot at approximately 0.6, and a blue dot at approximately 0.7. Shaded regions under the curve indicate the probability density for each point.

Table of contents

- [What is ShinyProxy?](#)
- [Why use it?](#)
- [Open Source](#)
- [Java Server Side](#)
- [Docker-based technology](#)
- [Open Source Shiny Package](#)

C'est quoi Shinyproxy ?

- Outil Open Source d'Open Analytics
- Écrit en Java
- Gère le déploiement d'applications avec docker (conçu à l'origine pour les apps Shiny)

Comment ça marche ?

Schématiquement :

- Chaque utilisateur en lancant l'application va finalement générer un conteneur
- Chaque visiteur a accès à son propre conteneur

Sur la machine qui fait tourner le tout :

- shinyproxy doit être installé et configuré
- Chaque application doit avoir sur la machine son image docker

Pourquoi choisir ShinyProxy?

En tant que dev de l'app :

- Chaque application a son propre environnement
 - Pas de problème de variables globales
 - Pas besoin d'optimisation avec {future} et {promises}
- La "home page" est créée nativement et est hautement personnalisable

En tant qu'admin :

- S'installe rapidement avec docker
- Maintenable par le développeur lui même.
 - | Cependant, shinyproxy demande plus d'effort pour déployer une application.

Ouvrir shinyproxy

- Direction <https://sp.cercvangirard.me/sp-votrenom>
- Même identifiant et mdp que Rstudio

The screenshot shows a dark-themed web interface for a Shiny proxy. At the top, there's a header bar with the Open Analytics Shiny Proxy logo, a user icon labeled 'cervan', and buttons for 'Admin' and 'Sign Out'. Below the header, a list of applications is displayed, with only one item visible: 'Hello Application'. A brief description below it states: 'Application which demonstrates the basics of a Shiny app'.

- Hello Application
Application which demonstrates the basics of a Shiny app

Comment configurer votre ShinyProxy ?

La configuration de SP se fait via un seul fichier le application.yml !

Regardons ensemble ce qu'il y a dans ce fichier !

Configuration de notre ShinyProxy

La config :

```
proxy:  
  title: Open Analytics Shiny Proxy  
  logo-url: https://www.openanalytics.eu/shinyproxy/logo.png  
  port: 8080  
  authentication: simple  
  admin-groups: admins  
  docker:  
    internal-networking: true
```

Configuration de notre ShinyProxy

Les users :

```
users:  
- name: jack  
  password: password  
  groups: admins  
- name: jeff  
  password: password
```

Configuration de notre ShinyProxy

Les apps :

```
specs:  
- id: 01_hello  
  display-name: Hello Application  
  description: Application which  
demonstrates the basics of a Shiny app  
  container-cmd: ["R", "-e",  
"shinyproxy::run_01_hello()"]  
  container-image:  
openanalytics/shinyproxy-demo  
  container-network: sp-example-net
```

La partie qui nous intéresse :

- id = donne l'id de l'app, peut être réutilisé pour charger l'app directement `/app/id`
- display-name, description = information sur la landing-page
- container-cmd = commande au lancant du conteneur (/!\ IMPORTANT)
- container-image = image qui sera lancé par le conteneur (/!\ IMPORTANT)
- container-network = configuration liée au déploiement de SP (ne pas modifier !)

Ajouter votre app !

Dans un premier temps, regarder votre formateur faire et puis on vous laissera le temps de le refaire :).

1- S'assurer d'avoir une image de son app {shiny}

- Si non, ajouter le dockerfile avec `golem::add_dockerfile_shinyproxy` et faire un build 2- Modifier le fichier `application.yaml` dans votre dossier {sp} 3- Relancer votre ShinuProxy avec le script `start.sh`



SP est hautement configurable, allons faire un tour sur le [site](#)

Bilan de la formation

Faisons le point

- Avons-nous répondu à la liste de vos besoins pour cette formation ?

Ressources

- Pour aller plus loin : "<https://rtask.thinkr.fr/blog/>"
- Pour aller plus loin : "thinkr.fr/blog/"

Alors ? Qu'en avez-vous pensé ?



**

Évaluation de la formation

Dans un soucis d'amélioration continue, nous recueillons l'impression de nos apprenants sur nos formations.

L'objectif pour nous est d'améliorer en permanence nos pratiques, pour pouvoir proposer des formations toujours plus qualitatives.

- Une évaluation à chaud à remplir maintenant
- Une évaluation à froid que vous recevrez dans 30 jours

Satisfaction

Avis Vérifiés™
Make internet safer

» L'authenticité des avis
» Transparence totale
» Tiers de confiance

Avis clients de Thinkr.fr

Calculé à partir de 118 avis obtenus sur les 12 derniers mois.*

★★★★★ 9.8 / 10

100% 118 avis 0% 0 avis 0% 0 avis

Evaluations clients

Voir les avis de mes amis

Avis client ★★★★★ 5 / 5

Très bonne formation. Merci

le 08/04/2021 par Catherine A.
suite à une expérience du 05/03/2021

Avis client ★★★★★ 5 / 5

La formation correspond exactement à la description. La contenu est à jour, agréable à consulter et directement applicable dans les projets avec R. Côté pédagogie nous étions suivi-es par des personnes à l'écoute et bienveillantes.

le 08/04/2021 par Gabriel C.
suite à une expérience du 12/03/2021

Avis soumis à un contrôle ⓘ

* 124 avis depuis le 07/07/2020

La gestion des avis clients par Avis Vérifiés de Thinkr.fr est certifiée conforme à la norme NF ISO 20488 "avis en ligne" et au référentiel de certification NF522 V2 par AFNOR Certification depuis le 28 Mars 2014. [En savoir plus](#)

Informations Thinkr.fr

DESCRIPTION : Spécialiste du langage R - Formation au logiciel R - Coaching d'équipe - Migration vers R - installation et configuration de serveur -

ADRESSE : ThinkR
50 rue arthur rimbaud
93300, aubervilliers

RECOMMANDER : [f](#) [t](#) [in](#) [g](#) [+](#)

WEBSITE : [thinkr.fr](#)

CATÉGORIE : Informatique et logiciels

CONTACT : Tel: 01 85 00 44 00

Vous serez probablement contactés d'ici quelques semaines par le site www.avis-verifies.com, merci de bien vouloir prendre les 30 secondes nécessaires pour laisser un témoignage.

Ces témoignages contribuent grandement à faire connaître nos formations.

Je n'ai pas le temps de pratiquer !

N'oubliez pas que quelle que soit la qualité d'une formation, pour être efficace et utile, il est nécessaire de pratiquer !

On vous propose de continuer à prendre 15 minutes régulièrement pour réfléchir à comment vous auriez fait un travail donné avec R grâce à vos toutes nouvelles connaissances en codage et en bonnes pratiques associées.

Lancez le chrono. Quand il s'arrête, vous vous arrêtez !

Si vous ne trouvez pas comment faire tout ce que vous auriez voulu cette fois-ci ce n'est pas grave car vous aurez déjà répondu à la pression avec vos méthodes de travail habituelles.

Comme ça pas de pression, et vous montez doucement en compétences au lieu d'oublier ce que vous aurez appris tout au long de votre formation !

Vous trouverez d'autres astuces de Lapin blanc sur [cette page de blog](#).

Cervan Girard

cervan@thinkr.fr