JDEV

INRAE

GitLab CI/CD | Environnement R

Retour d'expérience

10 juillet 2020 / JDEV2020 / Jean-François Rey

@jfrey_official

#JDEV2020

Qui sommes nous ?

Les présentations

INRAE - Institut national de recherche pour l'agriculture, l'alimentation et l'environnement

Centre PACA Avignon

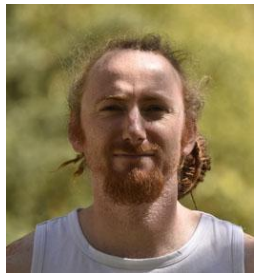
Unité **BioSP** (**Bio**statistique et **P**rocessus **S**patiaux) biosp.org

- Equipe recherche :
  - Statistique
  - Systèmes dynamiques
  - Ecologie-Épidémiologie

- Equipe Opérationnelle:
  - PESV (Plateforme D'Épidémiosurveillance en santé Végétale)

Loïc Houde
IE Info - NUMM
Sys et Réseaux

loic.houde@inrae.fr
http://loic.biosp.org



Jean-François Rey
IE Info - SPE
#DevOps

jean-francois.rey@inrae.fr
http://jeff.biosp.org

Introduction

Ingénierie logicielle et système (DevOps Culture)

Le développement et le partage de **productions informatiques** en recherche sont de plus en plus **nombreux** à l'unité BioSP. Ces derniers sont réalisés principalement par des non-informaticiens qui ne maîtrisent pas forcément les **processus de développement**.

Pour palier à la surcharge de tâches, au travail répétitif inutile des informaticiens et instaurer des "bonnes pratiques", plusieurs **solutions** ont été mises en place dont **l'automatisation** pour le **packaging** et la **livraison d'applications** en **continu**.

Ces solutions s'appuient sur des outils logiciels et une infrastructure matérielle.

Focus sur deux **pipelines GitLab** dans **l'écosystème R** :

- Pipeline d'intégration et de livraison de packages R sous différents OS
- Pipeline d'intégration et de déploiement d'applications R Shiny en production

Sommaire

- Technologies / Rappels
- Les Besoins
- Notre Solution
- Pipeline Package R
- Pipeline Apps R Shiny
- Super Pipeline
- Conclusion et perspectives

Technologies

Notions / Rappels rapides

L'outil central : GitLab CE

GitLab

En gros...

"GitLab is the first single application for the entire DevOps lifecycle." (Gitlab.com)

- Outil Web
- Basé sur Git (gestion du code / versionning)
- Planification et gestion de projets (Agile)
- Automatisation (**C**ontinuous **I**ntegration and **D**elivery / **D**eployment)
  - Via GitLab Runners
- Issues
- Docker registry
- Full Kubernetes
- ...

- MIT License (GitLab FOSS)
- Depuis 2011
- Dmitriy Zaporozhets and Valery Sizov

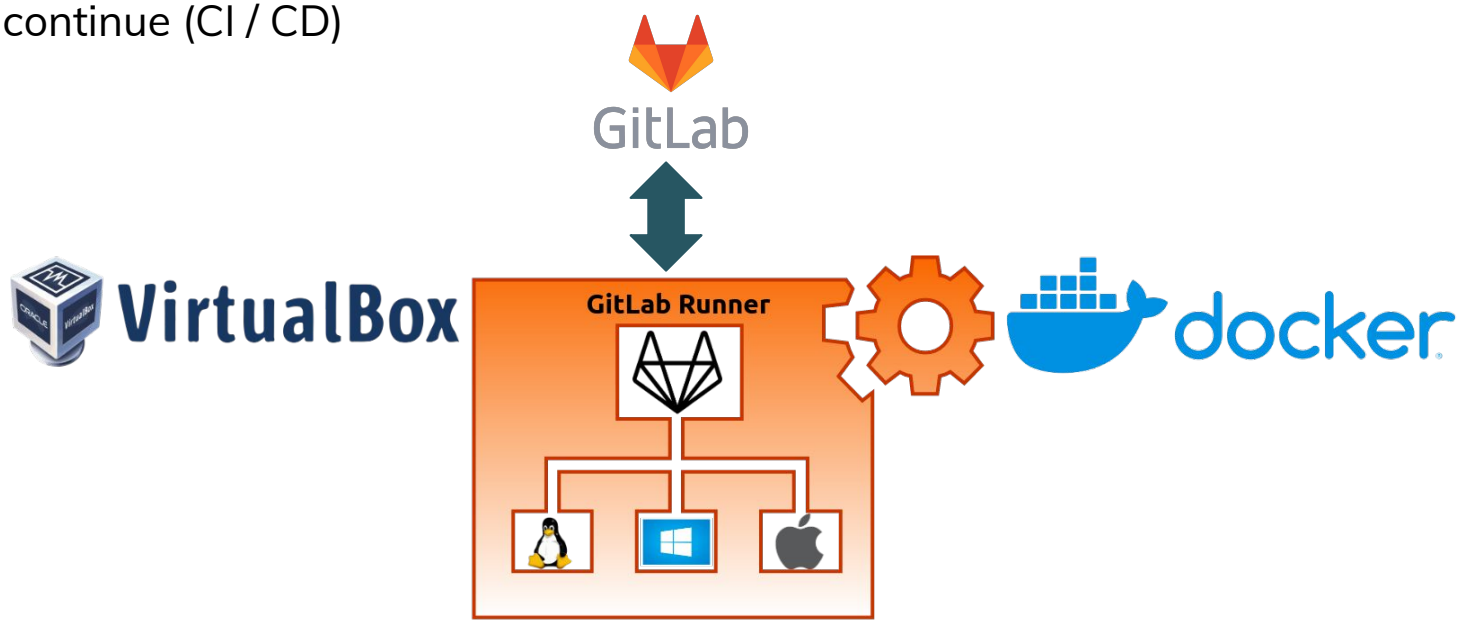
Like : GitHub, Travis, Jenkins, BitBucket, appveyor, R-hub... Mais dans **1** outil !

GitLab Runners

Ce sont eux qui bossent

- Ils font tourner les jobs (tâches demandées à GitLab)
- Via le fichier *.gitlab-ci.yml*
- Sur une multitude de machines et OS (physiques, virtuelles, laptop, ...)

Les Runners permettent l'intégration continue, le déploiement et la livraison continue (CI / CD)

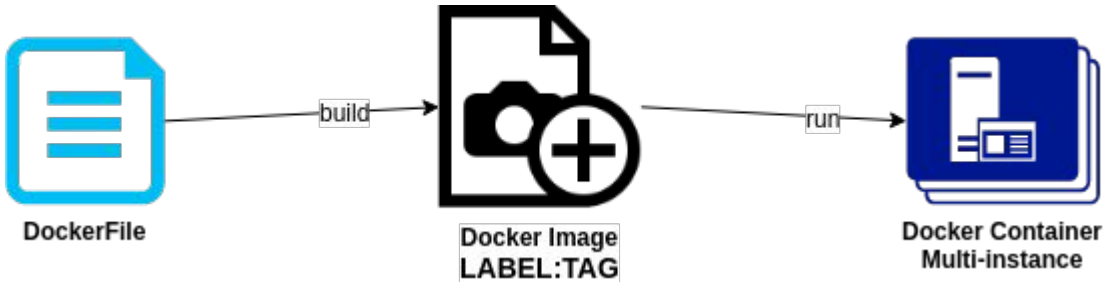


Docker

Docker est un logiciel libre qui permet de créer/lancer des applications dans des conteneurs logiciels (depuis 2013).

Un **container** est une forme de "~~Machine Virtuelle~~" plus légère, qui empaquette l'application et ses dépendances. Un **conteneur** tourne de manière isolé, il apporte une plus grande fiabilité et prédictibilité de l'application sur une grande variété de machines hôtes.

Une **image** est un "snapshot" d'un **container**. Une **image** est créée avec **Docker** (dockerfile) et elle produit un **container** quand elle est lancée.



Package R et App Shiny

R est un langage de programmation et un logiciel libre destiné aux statistiques et à la science des données soutenu par la R Foundation for Statistical Computing (Wikipédia).

Package R est le moyen le plus simple de partager des extensions pour **R**. En gros c'est un répertoire avec des conventions et normes.

- *DESCRIPTION*
- *NAMESPACE*
- *R/*
- *man/*
- *tests/*
- ...

Une **App R Shiny** est une application web interactive réalisée avec le package **Shiny** de RStudio. Elle fonctionne en local ou sur un serveur pour application Shiny (**ShinyServer**).

Les Besoins

Pourquoi ?

Les Besoins

Contexte

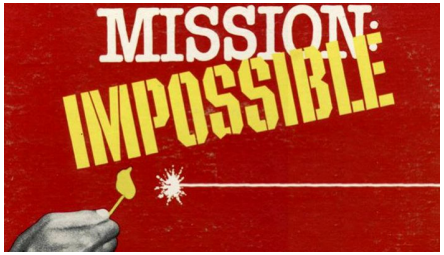
- ~15 Chercheurs **Statisticiens et/ou Mathématiciens**
  ~25 Ingés, doctorants, post-doc, CDD
  - réalisent des modèles Mathématiques et Statistiques
  - neophyte Dev informatique
  - code **R**
  - Collaboration Biologistes, écologues, botanistes, virologistes…
  - Collaboration international

- ~ 5 **Informaticiens** (2 CDD) - complémentaires
  - Systèmes et réseaux
  - Numéricien
  - Devs

- Divers projets plus ou moins avancés
  (amplitude variée, one shot, long terme)
  ANR, recherche interne…

Les Besoins

Ingénierie Informatique



- Développement d'outils de méthodes statistiques dans **l'environnement R**.
  - Centraliser les productions (codes et applications)
  - Partager facilement sous ≠ OS
  - Faciliter le **packaging R** et les soumissions au CRAN
  - Reproductibilité expérience
  - Dernièrement : publier facilement des applications **R Shiny**

- Faciliter et contrôler les projets informatiques.
  - Bonnes pratiques
  - Collaboratif (intra et extra INRAE)
  - Dépôts sécurisés privés et publics
  - Garder le contrôle des données
  - Documentations accessibles
  - Gestion de projet informatique
  - Outils simples d'utilisation et modernes
  - ...

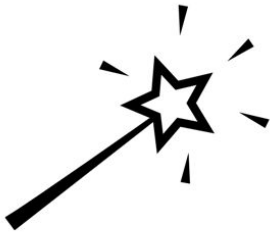
Bref, quelque chose de **simple** et **facile** à utiliser pour un **non-informaticien** mais répondant aux critères d'ingénierie pour un informaticien.

Notre Solution...

Comment ?

GitLab CE

Ingénierie Logicielle



Un outil **simple** et **moderne** : **GitLab CE** self hosted (FOSS)

- 2013-2014 : **https://gitlab.paca.inrae.fr**
- 2016 : **GitLab CI/CD** (compilations, tests, packaging et livraisons, ≠OS) Automatisé
- 2017 : **Pipeline R** (packaging et tests --as-cran) Automatisé
- 2019 : **Pipeline Apps Shiny** (dockerisation et déploiement) Automatisé

=> Se concentrer sur autre chose :

la recherche et le développement !

Ingénierie Matérielle

- Serveurs physiques en cluster (Hyperviseur Proxmox)
- Héberge les services (VM et container LXC)

Pipeline Package R

GitLab CI/CD et R CRAN architecture

Pipeline packaging R

Automatiser la création de packages R

Développement collaboratif d'un package R :

- Codeurs mais qui ne connaissent pas les pratiques de packaging
- Mainteneur : travail fastidieux et répétitif
- Codeurs et utilisateurs sous ≠ OS
- Code compilé (Fortran, C++...)
- Normes à respecter
- Partage et/ou soumission au CRAN parfois compliqué
- Pas de solution simple disponible

Centraliser, packager, tester et distribuer facilement !

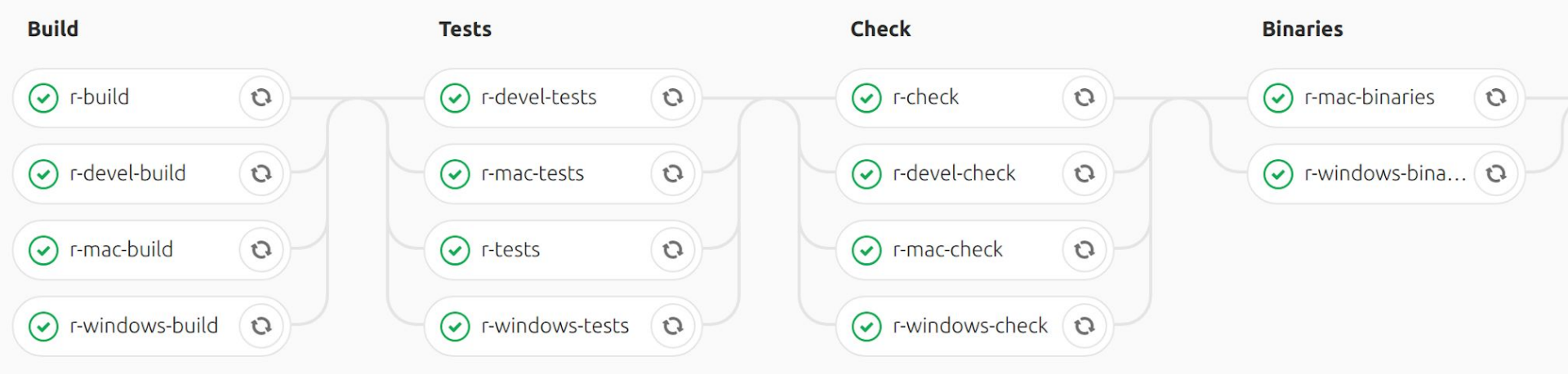
Le petit plus : l'environnement R se prête bien au pipeline grâce aux outils R déjà disponibles qui nous facilitent la vie (Roxygen2, devtools, Renv...).

Pipeline packaging R

Automatiser la création de packages R

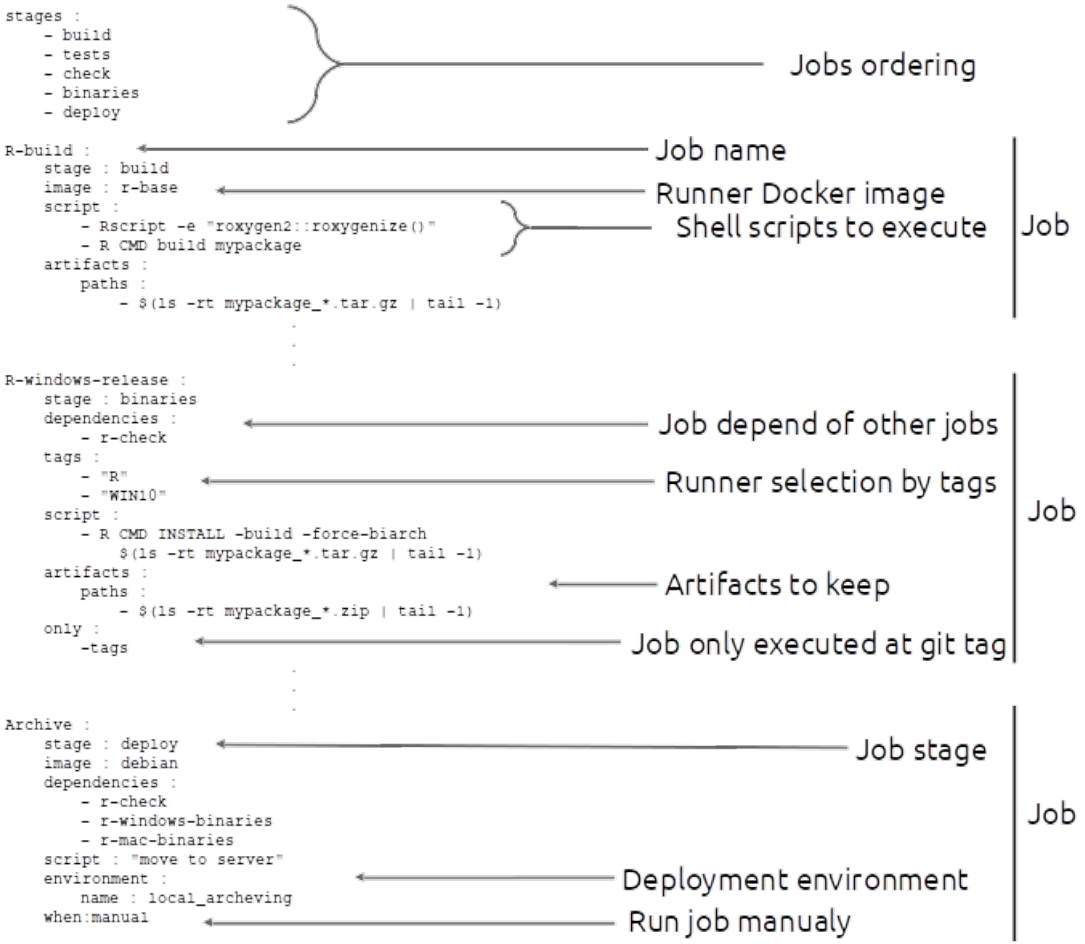
On s'appuie sur GitLab CI/CD et les outils R.

1. Dépôt versionné du code du package (GitLab)
2. Instructions pour générer et tester le package (**.gitlab-ci.yml**)
3. Réaliser toutes les tâches (GitLab Runners)
4. Rapports, logs, packages sources et binaires (GitLab)



Pipeline packaging R

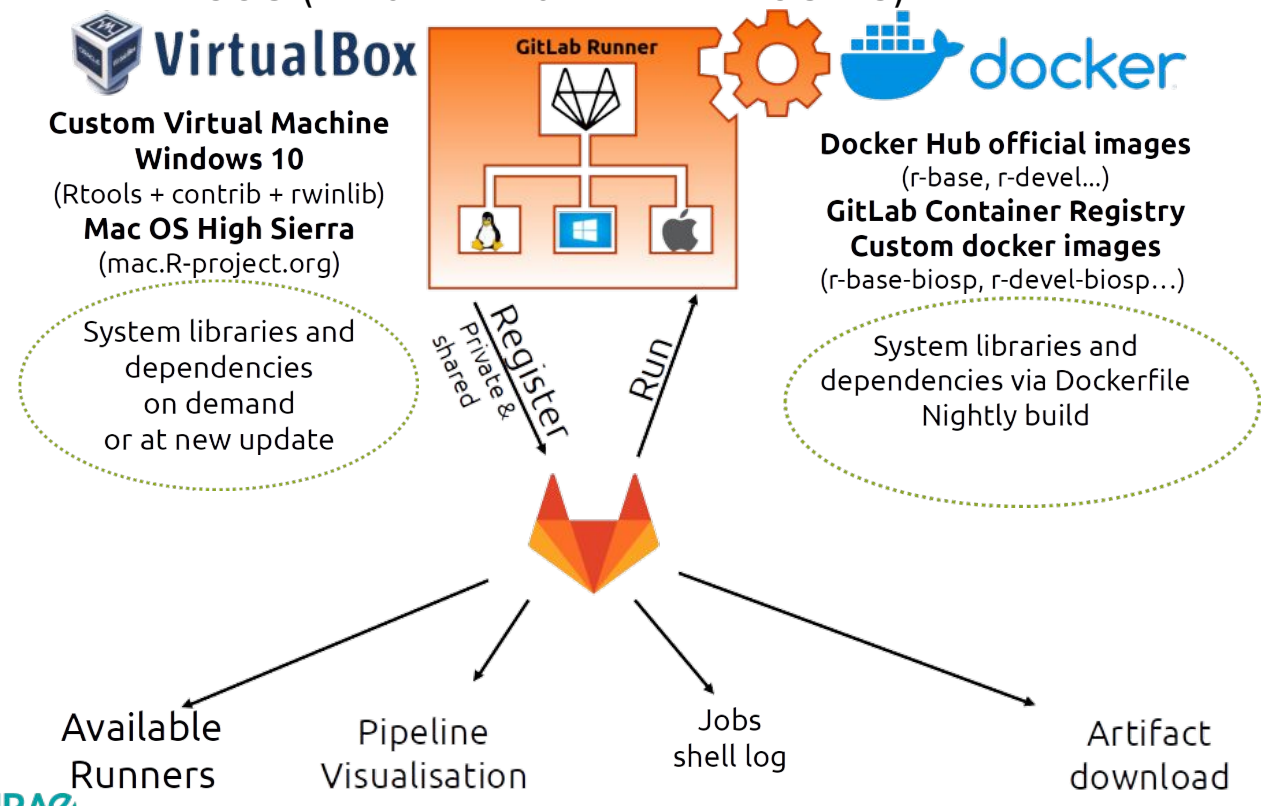
.gitlab-ci.yml



Pipeline packaging R

4 Runners disponibles sous différentes versions

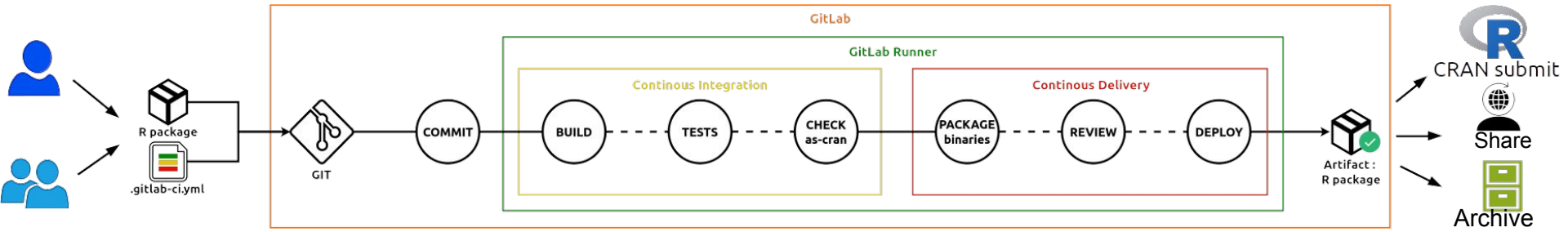
- **R** (version actuelle sous docker nightly-build)
- **R-devel** (version en dev sous docker nightly-build)
- **R-win** (V3.6 et V4.0 sous VM Windows 10)
- **R-MACOS** (V3.6 et V4.0 sous Mac OS HS)



Pipeline packaging R

Architecture

- **Dépôt (projet) GitLab :** “versionne” le code du package
- **.gitlab-ci.yml :** automatisation des builds et tests (CI/CD)
- **Runners :** exécutent toutes les tâches (jobs)
- **Artifacts** : production des taches (package R valide)



Exemples :
https://gitlab.paca.inrae.fr/r-ecosystem/cookbooks/r-packages-ci-cd

Pipeline packaging R

En résumé

- **Mainteneur**
  - Mise en place du pipeline 1 fois (**.gitlab-ci.yml**)
  - Plus besoin de jongler avec ≠ VM et OS (**Runners**)
  - Donne du **temps** pour faire autre chose
  - Plus besoin d'un tiers ou de solutions externes
  - Réduit / Supprime les erreurs de soumission au CRAN

- **Développeurs**
  - Pas besoin de savoir faire un package R
  - Développement collaboratif
  - Erreurs qui remontent plus vite
  - Distribution du package en DEV (source et binaires)
  - Livrable plus régulier

- **Package R**
  - Code centralisé, versionné, normé
  - Pérennité du package (CRAN et hors CRAN)
  - Partage facilité aux utilisateurs

Pipeline Apps R Shiny

GitLab CI/CD, Docker et Apps R Shiny

Pipeline Apps R Shiny

Automatiser la création et la publication d'une App. R Shiny

- **Contraintes**
  - Développement à plusieurs (non-informaticiens)
  - Serveur single-user (version libre)
  - Fortes dépendances des Apps aux versions des packages et de R
  - Trop fortes dépendances en production (ressources, bibliothèques systèmes...)

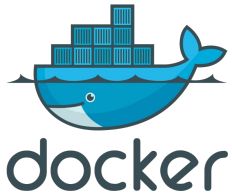
- **On veut**
  - du code centralisé,
  - une application isolée, single-user et/ou multi-user,
  - effectuer des tests,
  - avoir un déploiement en production facile !
  - Facile pour les Devs

=> GitLab CI/CD + Docker + ShinyServer

Pipeline Apps R Shiny

Solution serveur

- **Docker**
  - Applications isolées
  - Fin des dépendances systèmes



1 App R Shiny (+ 1 serveur Shiny) + 1 Dockerfile
=>
1 image docker de l'App

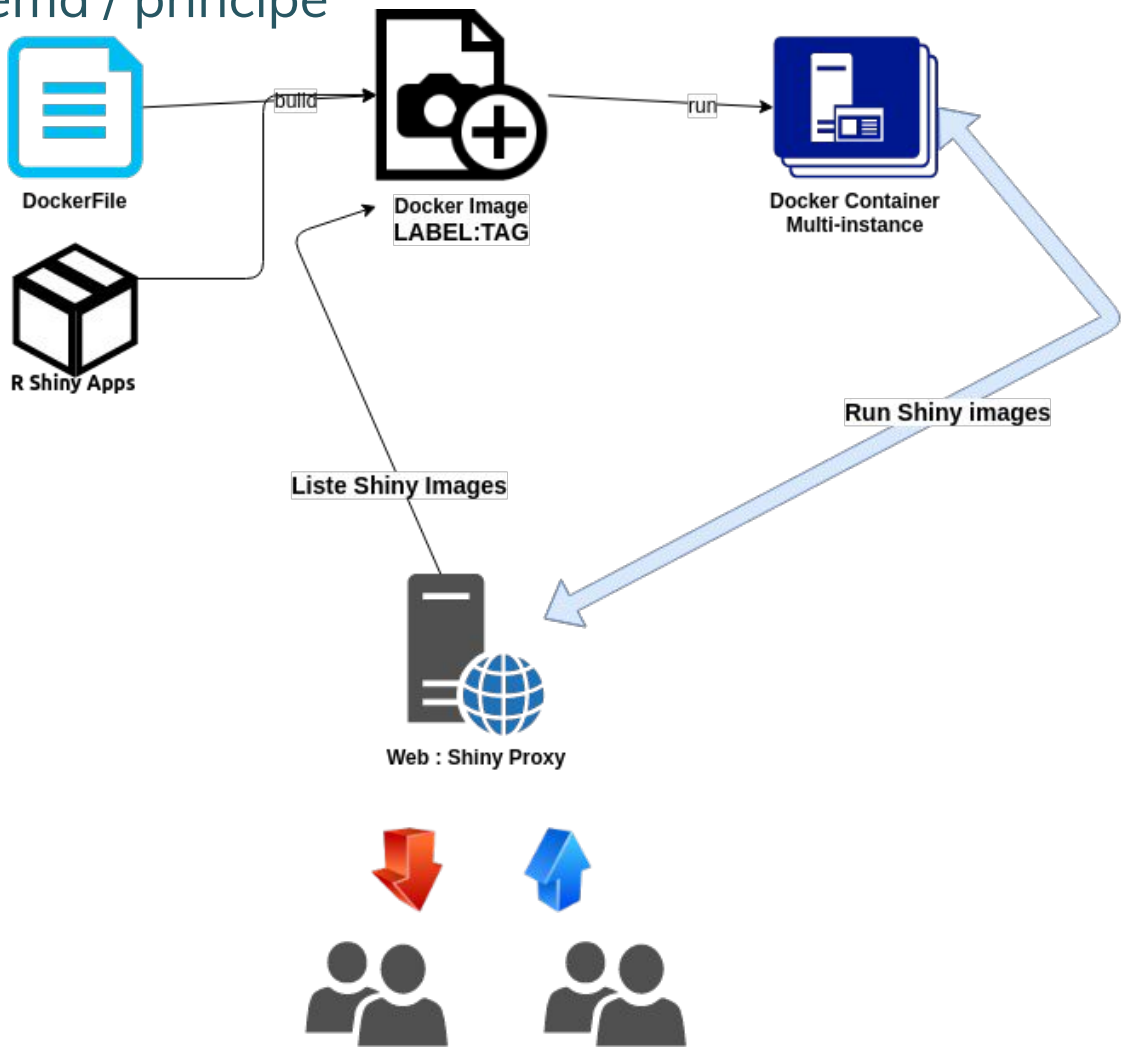
1 image de l'App
=>
Multi instances de Containers

- **ShinyProxy** est une application serveur qui permet de déployer de manière transparente des Apps Shiny
  - Liste des applications (images Docker)
  - Gestion des applications (ordonnanceur de containers)

- Public : https://shiny.biosp.inrae.fr
  - ➢ accès direct : shiny.biosp.inrae.fr/app_direct/<appID>

Pipeline Apps R Shiny

Schéma / principe



Pipeline Apps R Shiny

GitLab CI/CD dans tout cela

- **A la main à chaque fois :**
  - Erreurs, oublis (“Comment j’ai fait la dernière fois déjà ?”)
  - Prend du temps actif et inactif
  - Question de stockage (“Où c’est déjà sur mon ordi ?”)
  - “L’informatique c’est pas pour moi !”
  - “C’est qui qui s’en occupe ? Il est où super-admin ?”
  - …

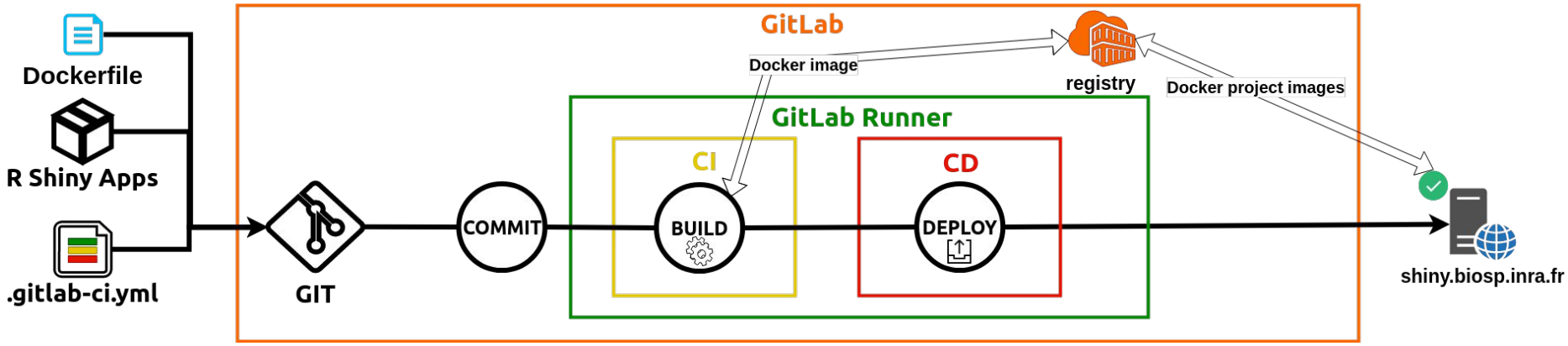
GitLab CI/CD simplifie tout le processus + accès aux fonctionnalités (rollback, reproductibilité, partage, sécurité…)

- **Centraliser** l’application **Shiny**, le **Dockerfile**, et les **images** Docker
- **Générer** et **stocker** l’image docker de l’App Shiny **automatiquement**
- Cacher les **données sensibles** (mdp, ID…)
- **Déployer automatiquement** les mises à jour en Prod

Pipeline Apps R Shiny

Architecture

- **Dépôt (projet) GitLab** : App Shiny + Dockerfile
- **.gitlab-ci.yml** : automatisation du build (dind) et déploiement
- **Registry** : Gestion des images docker par dépôt



Exemple :
https://gitlab.paca.inrae.fr/r-ecosystem/cookbooks/r-shiny-app-ci-cd

Pipeline Apps R Shiny

En résumé

- **Informaticiens**
  - Centralisent le code de l'App (GitLab)
  - Mise en place du pipeline 1 fois (.gitlab-ci.yml)
  - Fin des contraintes systèmes ou R (images docker)
  - Du temps libéré pour faire autre chose

- **Développeurs R**
  - Code à jour -> App en Prod à jour
  - Rien à gérer hormis le code de l'App
  - Moins d'erreurs ou d'oublis
  - Plus de temps pour développer
  - Facile à utiliser (c'est transparent)

- **Serveur**
  - Moins d'administration
  - Multi-user "scalable"
  - Apps sécures, stables et pérennes

- **Utilisateurs**
  - Apps à jour
  - Facile d'accès (via docker ou serveur web)

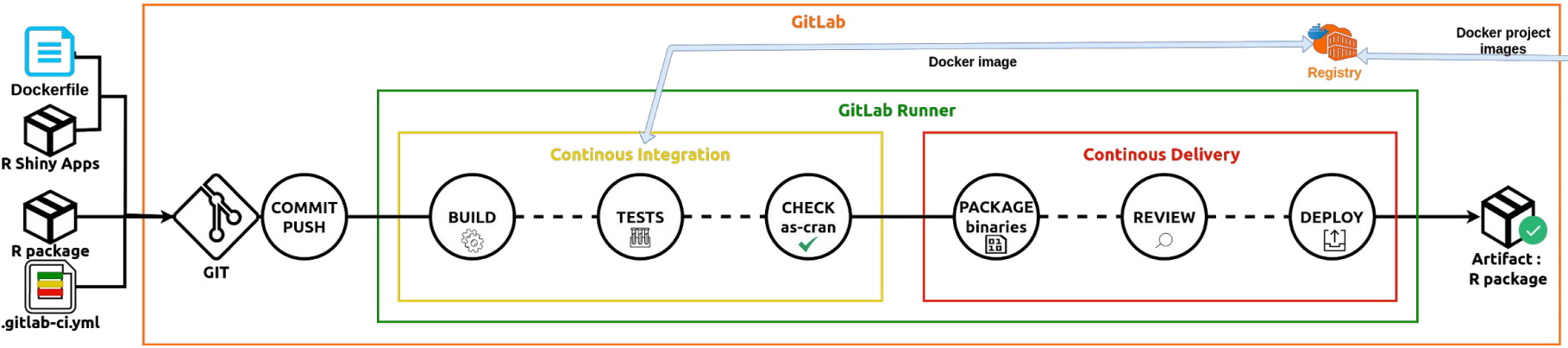
Super Pipeline

Les deux pipelines réunis

GitLab CI/CD et environnement R

Package R avec une App R Shiny

- **Dépôt (projet) GitLab** : code Package R + App Shiny + Dockerfile
- **.gitlab-ci.yml** : automatisation du build, tests, livraison et déploiement
- **Runners :** exécutent toutes les tâches (jobs)
- **Registry** : gestion des images docker par dépôt
- **Artifact** : package R valide + App R shiny image Docker



Exemple : https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi

https://shiny.biosp.inrae.fr/app/landsepi

Conclusion

En bref

En conclusion

Pipeline GitLab CI/CD et environnement R

- A changé la vie des chercheurs et des informaticiens
  - **Les chercheurs**
    Autonomes
    Codent et voient l'effet immédiatement
    Partagent facilement
    Reproductibilité
  - **Les informaticiens**
    Moins de travail répétitif
    Moins de bugs à gérer
    Moins d'administration système
    Plus de disponibilité

=> Mise en place des pipelines : investissement en temps plus rentable
=> Gains en productibilité, réactivité, fréquence de livraison, qualité

- Quelques (petits) chiffres :
  - ~ 30-35 projets depuis 2017 avec pipeline
  - ~ 2000 jobs/ans de 5min à 2h

En conclusion

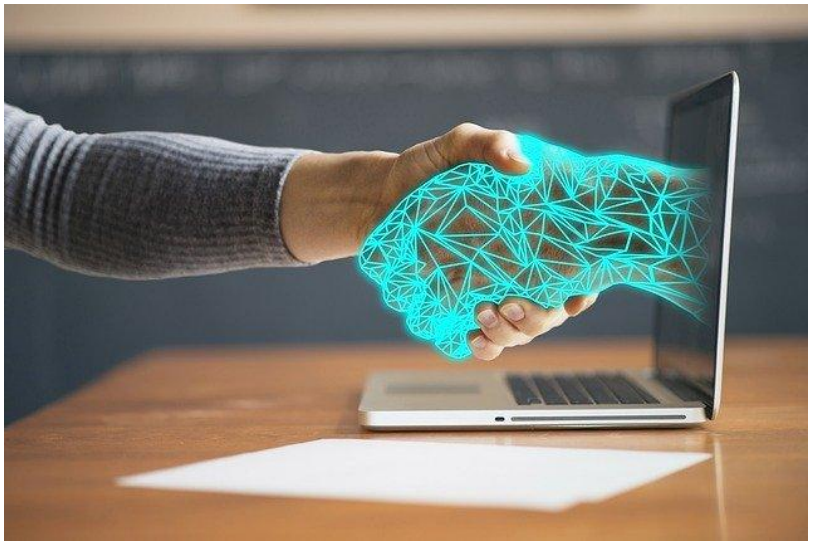
Pipeline GitLab CI/CD et environnement R

- **1** outil de gestion de projet
- **Centralisation** du code, de la doc et des artifacts
- **Simple et facile** à utiliser pour un néophyte
- On s'occupe du **code**, le **reste c'est automatique CI/CD**
  - **GitLab Runners** Linux, Windows et Mac OS
  - **Docker** registry
- Pipelines utilisables par tous les **collaborateurs**
- Pipelines qui facilitent le **partage** de **packages binaires R** et les **soumissions** au **CRAN**
- Pipelines qui facilitent la vie des **Apps R Shiny** plus opérationnelles

- **Perspectives :**
  - Génération et Provisioning des Runners VMs (**IasC**)
  - Dev ShinyProxy back office
  - K8S en prod
  - ...

Merci !

Questions ?



Idées ?

@jfrey_official

Références

1. GitLab : https://about.gitlab.com/
2. GitLab-Runner : https://docs.gitlab.com/runner/
3. GitLab Registry : https://docs.gitlab.com/ee/user/packages/container_registry/
4. Docker : https://docker.com
5. VIrtualBox : https://www.virtualbox.org/
6. R : https://www.r-project.org/
7. R Shiny : https://shiny.rstudio.com/
8. Shiny Server : https://rstudio.com/products/shiny/shiny-server/
9. ShinyProxy : https://www.shinyproxy.io/
10. Groupe ecosystème R : https://gitlab.paca.inrae.fr/r-ecosystem
11. Cookbook Gitlab / R : https://gitlab.paca.inrae.fr/r-ecosystem/cookbooks
12. Serveur Shiny public : https://shiny.biosp.inrae.fr/