Gérer des données dans GeoServer avec la librairie R geosapi

Intervenant: Emmanuel Blondel – Consultant SI
Contact: emmanuel.blondel1@gmail.com

Atelier R métadonnées - Agropolis, Montpellier (France) 8 - 9 Février

Introduction

- Langage R
  - Large communauté d'utilisateurs: gestionnaires de données, scientifiques, statisticiens
  - Pas limité à une audience d'informaticiens et de développeurs
  - Intérêt grandissant au delà du traitement statistique:
    - automatisation de tâches de gestion de données au sens large (conversion, traitement / analyses thématiques)
    - Gestion, traitement et analyse de données spatialisées
- GeoServer
  - Fournit un API pour la gestion du catalogue de données de Geoserver
- Outils programmatiques pour la publication de données
  - Java → geoserver-manager
  - Python → gsconfig

Introduction

- Construire progressivement un interface R pour l'API GeoServer
- Références
  - http://geoserver.org/
  - https://github.com/geosolutions-it/geoserver-manager
- Avantages
  - API fonctionnel pour la gestion des ressources du catalogues (Création, Lecture, Mise à jour, Suppression): espaces de travail, dépôts de données, couches, featureTypes, styles, etc
  - Facile d'utilisation en combinaison avec geometa et geonapi pour exposer des données sur des services OGC et les référencer dans des fiches de métadonnées.
- Inconvénients
  - API propre au modèle GeoServer

Geosapi – le projet

- FOSS (Free and Open Source Software)
- Page web du projet: https://github.com/eblondel/geosapi
- Statut actuel des développements:
  - Librairie disponible sur CRAN: https://cran.r-project.org/package=geosapi
  - Documentation exhaustive des codes disponible via R, et progressivement sur le wiki
- Perspectives de développements
  - Ajout de classes et méthodes manquantes
  - Gestion de nouvelles fonctionnalités Geoserver
- Recherche de financements (sponsors et projets techniques)
- Formations techniques

Geosapi – Comment ça marche?

- Modèle R orienté “objet” (modèle basé sur la librairie “R6”)
- Ensemble de **classes** et de **méthodes** (“fonctions”) associées qui sont accessible via le signe “$”.
- Principe simple d’utilisation:

  1. On crée un objet d’une **classe** donnée avec la méthode **new()**:

  ```
  monObjet <- MaClasse$new()
  ```

  2. Puis on accède aux différentes méthodes via le “$”. Exemple: méthode pour spécifier le nom d’un objet

  ```
  monObjet$setName("monnom")
  ```

Geosapi – Chargement de la librairie

- Installer et charger le dernier *release* stable de geosapi

```
install.packages("geosapi")
library(geosapi)
```

- Installer et charger la version de geosapi en cours de développement à partir du dépôt source Github (nécessite la librairie *devtools*)

```
library(devtools)
install_github("eblondel/geosapi")
library(geosapi)
```

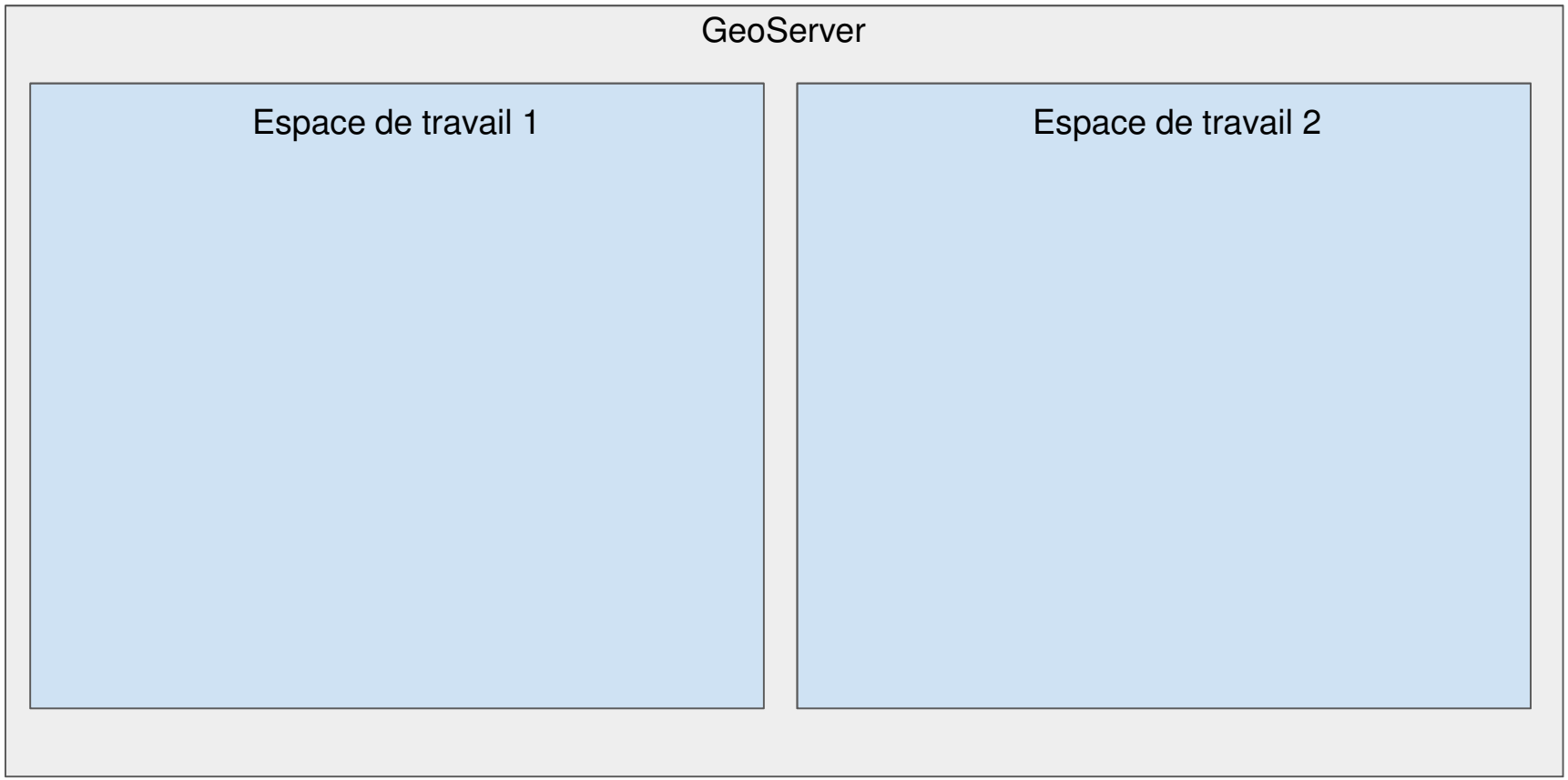
Geosapi – Pratique - GSManager

- La classe **GSManager**, ou “Gestionnaire GeoServer” est le point d’entrée unique pour se connecter à GeoServer et effectuer des opérations.
- Paramètres à spécifier:
  - URL de GeoServer
  - Utilisateur / Mot de passe
  - Logger: NULL par défaut (aucun log), INFO: logs geosapi seulement, DEBUG: logs geosapi et CURL (logs complets des opérations)

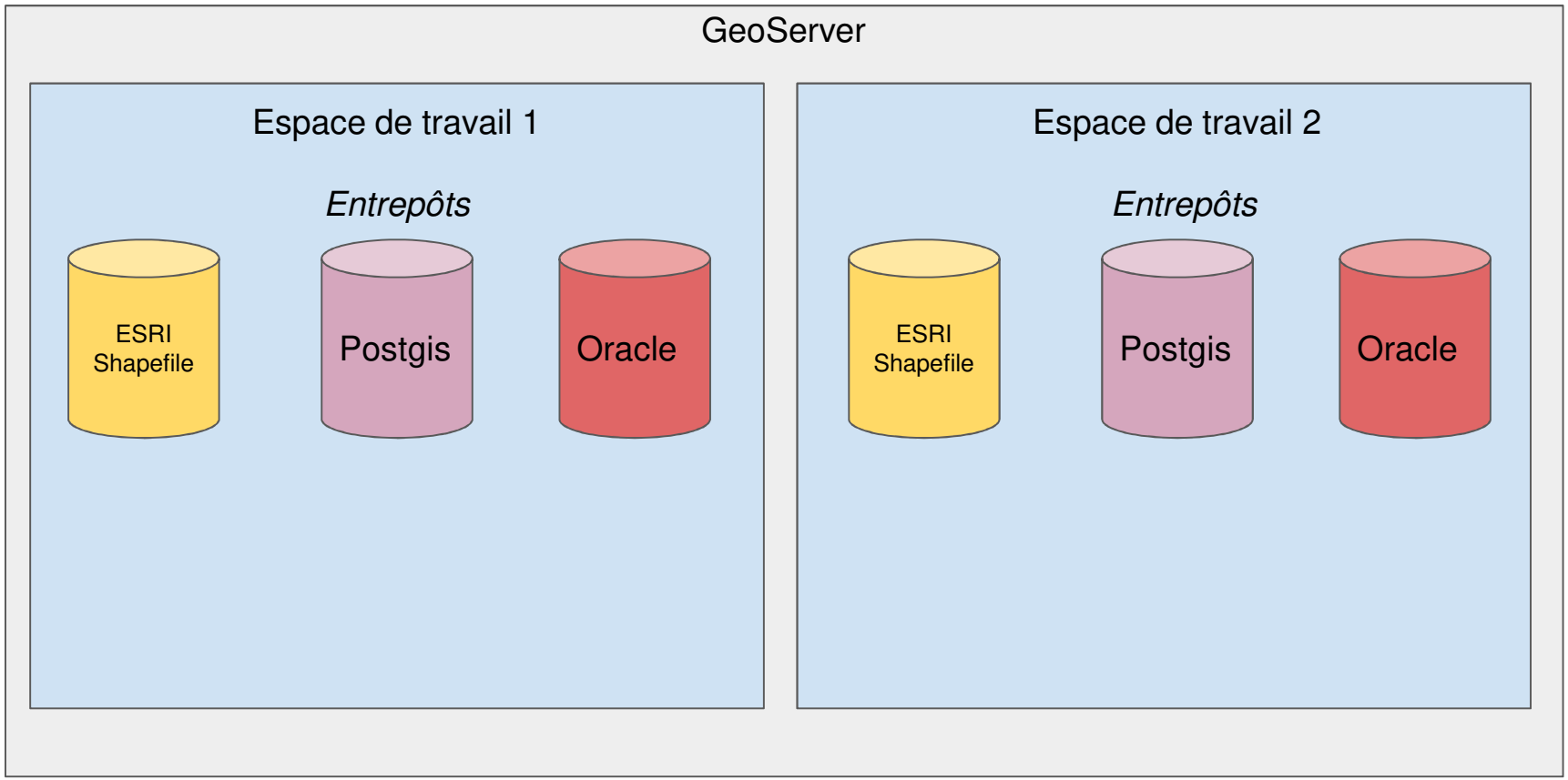
```
GS <- GSManager$new(
  url = "http://localhost:8080/geoserver",
  user = "admin",
  pwd = "geoserver",
  logger = "DEBUG"
)
```

Geosapi – Pratique - Les entités GeoServer en résumé

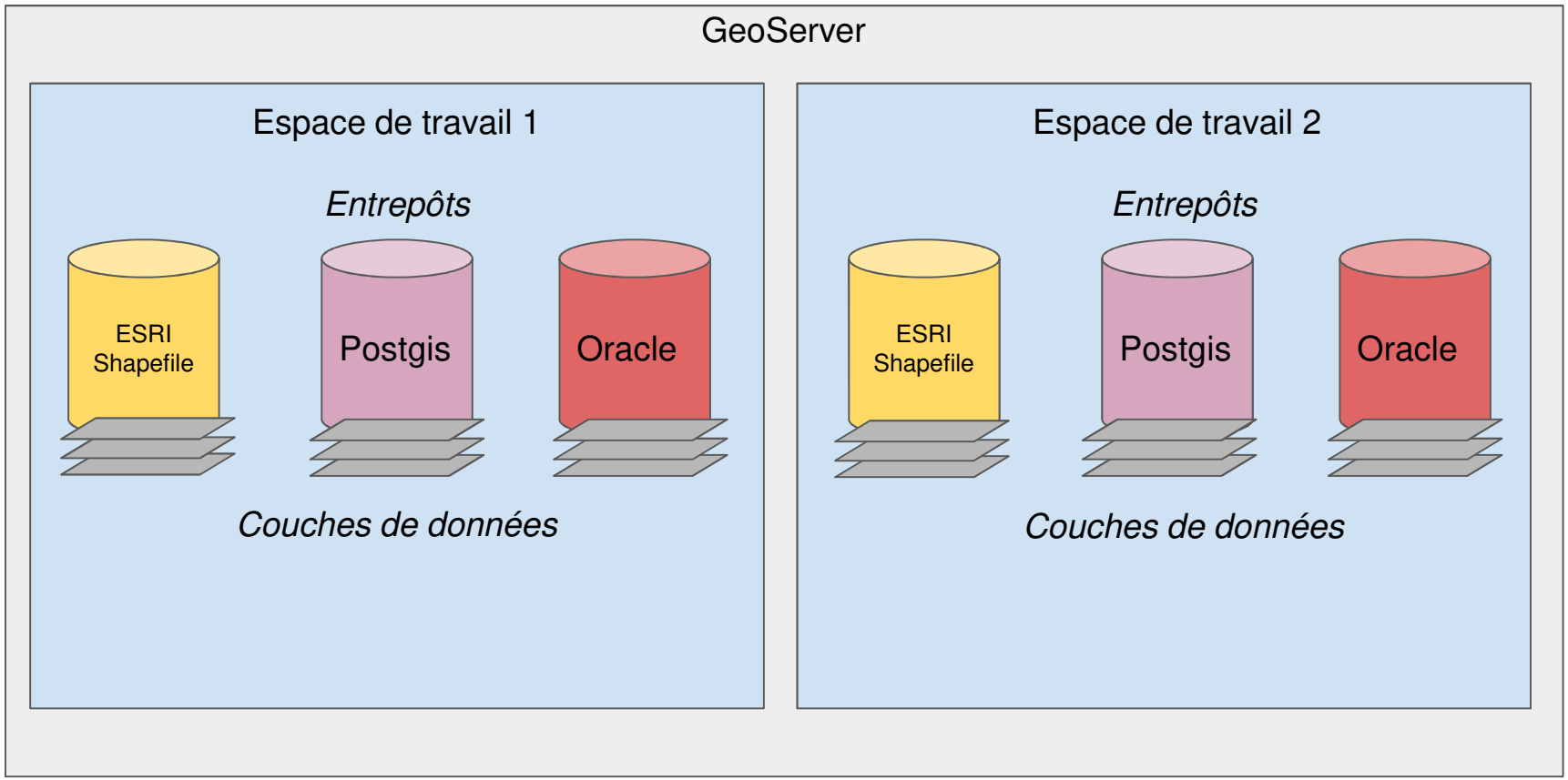
Geosapi – Pratique - Les entités GeoServer en résumé



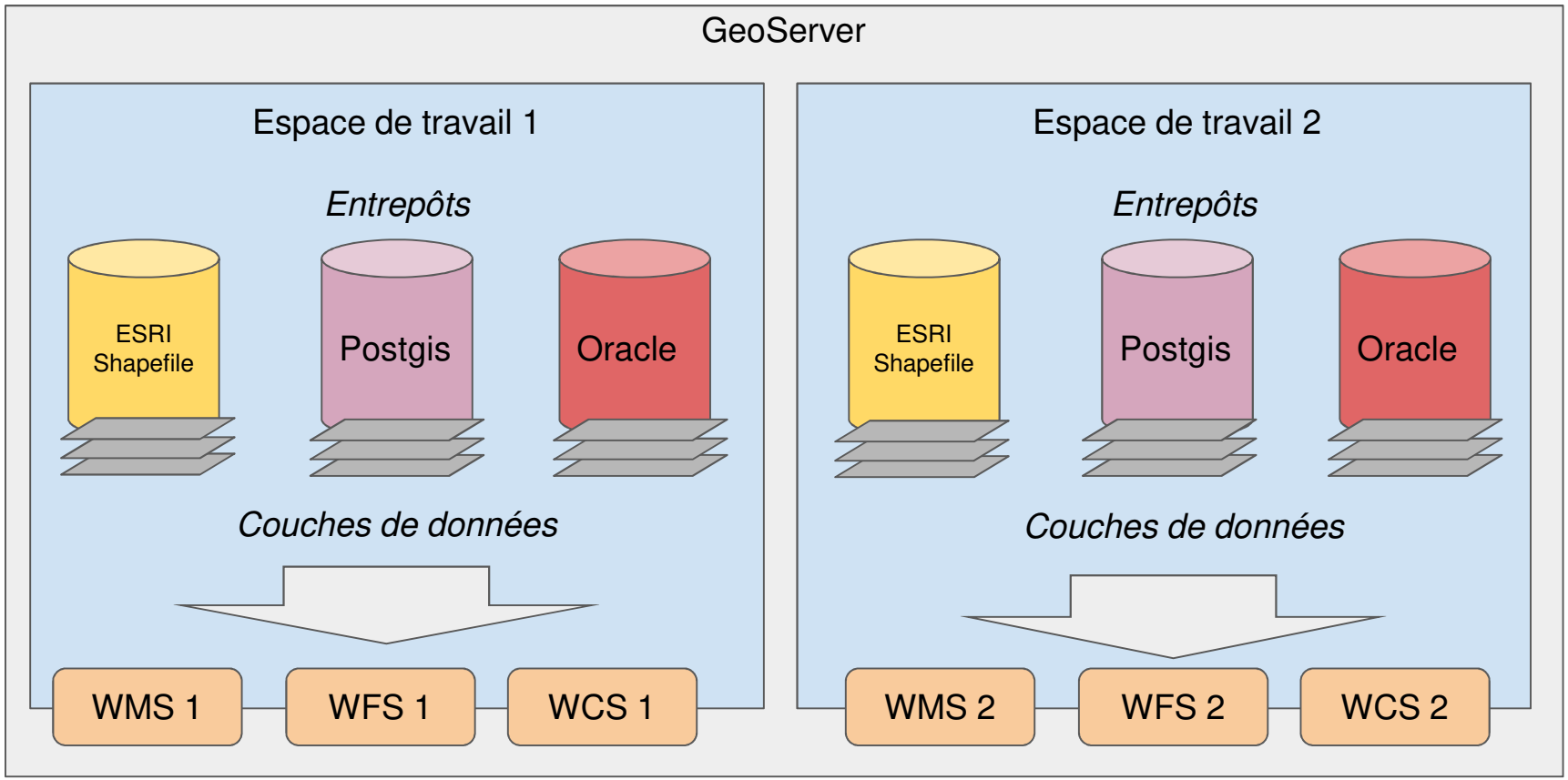
Geosapi – Les entités GeoServer en résumé



Geosapi – Les entités GeoServer en résumé

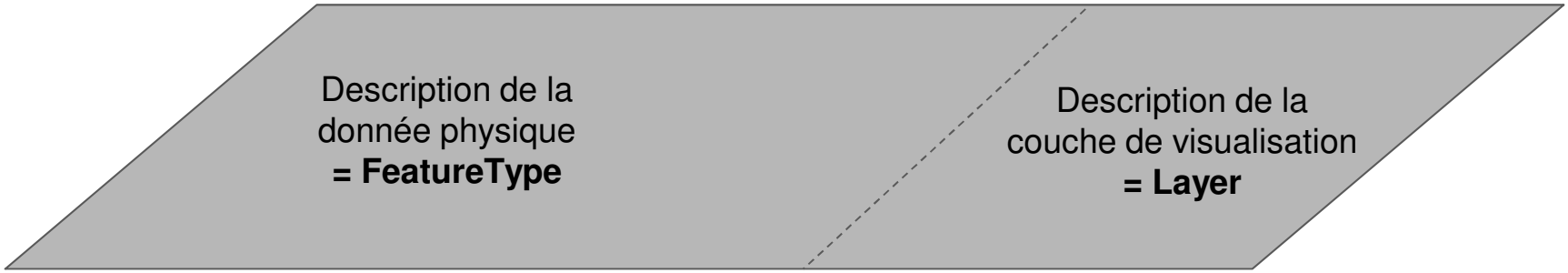


Geosapi – Les entités GeoServer en résumé



Geosapi – Pratique - Les entités GeoServer en résumé

Ressource de données GeoServer = 2 entités différentes!



- Nom, Titre, Abstract, mots-clés
- Spatial Reference System
- BoundingBox
- Dimensions
  - Standard (TIME, ELEVATION)
  - Vue dynamique (paramétrisée ou pas)

- Styles (SLD) associés à la couche
  - Style par défault
  - Styles disponibles pour la couche
- Statut de publication WMS

Geosapi – Pratique

- Pré-requis: se connecter à un GeoServer

```
#se connecter au GeoServer
GS <- GSManager$new(
  url = "http://localhost:8080/geoserver",
  user = "admin", pwd = "geoserver",
  logger = "DEBUG"
)
```

- Objectif: Apprendre à publier des ressources dans GeoServer

Geosapi – Pratique - Gérer un espace de travail

- Avantage de l'espace du travail:
  - Permet de grouper / catégoriser de ressources de données. Exemples:
    - Par projet?
    - Par fournisseur de données?
    - Par niveau d'agrégation?
    - Données confidentielles vs. données publiques
    - … (c'est à vous décider la manière de grouper vos données!)
  - 1 espace de travail → 1 point d'entrée de services OGC (WMS/WFS/WCS).
- Documentation sur le wiki

- Créer un espace de travail personnel avec mon nom (minuscules)

```
#je crée mon espace de travail
ws_name <- paste("sdilab","monnom", sep = "_")
```

```
created <- GS$createWorkspace(ws_name, "http://monnom")
```

Geosapi – Pratique - Gérer un espace de travail

- Actualiser un espace de travail? Relativement limité → namespace

```
#j'actualise mon espace de travail
updated <- GS$updateWorkspace(ws_name, "http://monnouveaunom")
```

- Supprimer un espace de travail
  - Supprimer toutes les ressources liées à cet espace? `recurse = TRUE`

```
#je supprime mon espace de travail
deleted <- GS$deleteWorkspace(ws_name, recurse = TRUE)
deleted
```

- Liste des espaces de travail?

```
 #get workspace names only
  wsnames <- GS$getWorkspaceNames()
```

Geosapi – Pratique - Gérer un entrepôt Shapefile

- Entrepôt simple: 1 entrepôt = 1 shapefile (1er entrepôt géré dans Geoserver!)

```
#nom de mon entrepôt
ds_name <- paste(ws_name, "shapefiles", sep = "_")

#objet décrivant mon entrepot
datastore <- GSShapefileDataStore$new(
   dataStore= ds_name, description = "mon shapefile",
   enabled = TRUE, url = "file://data"
)

#je crée mon datastore dans mon espace de travail
created <- GS$createDataStore(ws_name, datastore)
created
```

Geosapi – Pratique - Gérer un entrepôt Shapefile

- Actualiser un entrepôt?

```
ds <- GS$getDataStore(ws_name, ds_name)
ds$setDescription("une autre description")
ds$setEnabled(FALSE) #désactiver l'entrepôt?
updated <- GS$updateDataStore(ws_name, ds)
```

- Supprimer un entrepôt
  - Supprimer toutes les ressources (couches de données) liées à cet entrepôt? `recurse = TRUE`

```
deleted <- GS$deleteDataStore(ws_name, ds_name, recurse = TRUE)
deleted
```

- Liste des entrepôts?

```
#get workspace names only
dsnames <-GS$getDataStoreNames(ws_name)
```

Geosapi – Pratique - Publier un Shapefile?

- Publier un Shapefile dans Geoserver à partir de R ?
  - Uploader le shapefile
  - Créer la ressource associée (featureType + layer)
- Etape 1: Upload
  - ‘configure’: “none” → faire uniquement l’upload et ne pas configurer de couche automatiquement
  - ‘update’: “append” or “overwrite”

A faire télécharger le dossier “shapefile” du workspace VRE, et le charger dans RStudio....

```
#télécharger dans RStudio le shapefile zippé
setwd("~/shapefile") #le dossier qui s’est créé lors du chargement du zip
zipfilename = "resource_all_points.zip"
zip(zipfilename, files = list.files(pattern="resource_all_points"))

#upload du shapefile
uploaded <- GS$uploadShapefile(
  ws_name, ds_name, endpoint = "file",configure = "none",
  update = "overwrite", zipfilename, "UTF-8"
)
```

Geosapi – Pratique - Publier un Shapefile?

- Etape 2: Configurer la ressource de données
  - 2. 1. Configurer le feature type

```
mylayername <- paste(ws_name, "layer", sep = "_")
featureType <- GSFeatureType$new()
featureType$setName(mylayername)
featureType$setNativeName("resource_all_points")
featureType$setAbstract("abstract")
featureType$setTitle("title")
featureType$setSrs("EPSG:4326")
featureType$setNativeCRS("EPSG:4326")
featureType$setEnabled(TRUE)
featureType$setProjectionPolicy("REPROJECT_TO_DECLARED")
featureType$setLatLonBoundingBox(-180,-90,180,90, crs = "EPSG:4326")
featureType$setNativeBoundingBox(-180,-90,180,90, crs ="EPSG:4326")
md1 <- GSMetadataLink$new(type = "text/xml", metadataType = "ISO19115:2003", content = "http://somelink.org/xml")
featureType$addMetadataLink(md1)
md2 <- GSMetadataLink$new(type = "text/html", metadataType = "ISO19115:2003", content = "http://somelink.org/html")
featureType$addMetadataLink(md2)
created <- GS$createFeatureType(ws_name, ds_name, featureType)
```

Geosapi – Pratique - Publier un Shapefile?

- Etape 2: Configurer la ressource de données
  - 2. 2. Configurer le layer

```
layer <- GSLayer$new()
layer$setName(mylayername)
layer$addStyle("generic")
created <- GS$createLayer(layer)
```

Geosapi – Pratique - Publier un Shapefile?

- Services associés
  - WMS: http://localhost:8080/geoserver/sdilab_monnom/wms?service=WMS&version=1.1.0&request=GetMap&layers=sdilab_monnom:sdilab_monnom_layer&styles=&bbox=-180.0,-90.0,180.0,90.0&width=768&height=384&srs=EPSG:4326&format=application/openlayers
  - WFS: http://localhost:8080/geoserver/sdilab_monnom/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=sdilab_monnom:sdilab_monnom_layer
- A quoi ressemble le WMS/WFS *GetCapabilities* (description des services)
  - WMS: http://localhost:8080/geoserver/sdilab_monnom/ows?service=wms&version=1.3.0&request=GetCapabilities
  - WFS: http://localhost:8080/geoserver/sdilab_monnom/ows?service=wfs&version=1.1.0&request=GetCapabilities

Merci pour votre attention