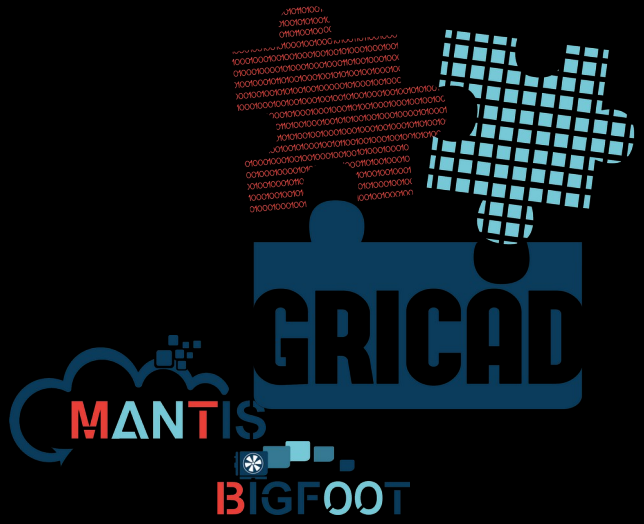





# Utilisation avancée d'iRods dans le projet Orchamp-vision

Bruno Bzeznik, Elias Chetouane, Vincent Miele, Julien Renaud, Wilfried Thuiller

# Le projet ORCHAMP

- Observatoire spatio-temporel de la biodiversite et du fonctionnement des socio-ecosystemes de montagne: observer, analyser et modéliser les changements environnementaux sur la base de multiples protocoles de suivi
- Dans le cadre de la présente collaboration GRICAD/LECA, on s'intéresse à 2 aspects du projet ORCHAMP:
- Orchamp-vision
  - Pièges photo
- Orchamp-sound
  - Enregistrements accoustiques

https://orchamp.osug.fr/

# Orchamp vision

Des “pièges photo”, offline (caméras avec détection de mouvement) sont disposés sur les “gradients” Orchamp

# Orchamp vision

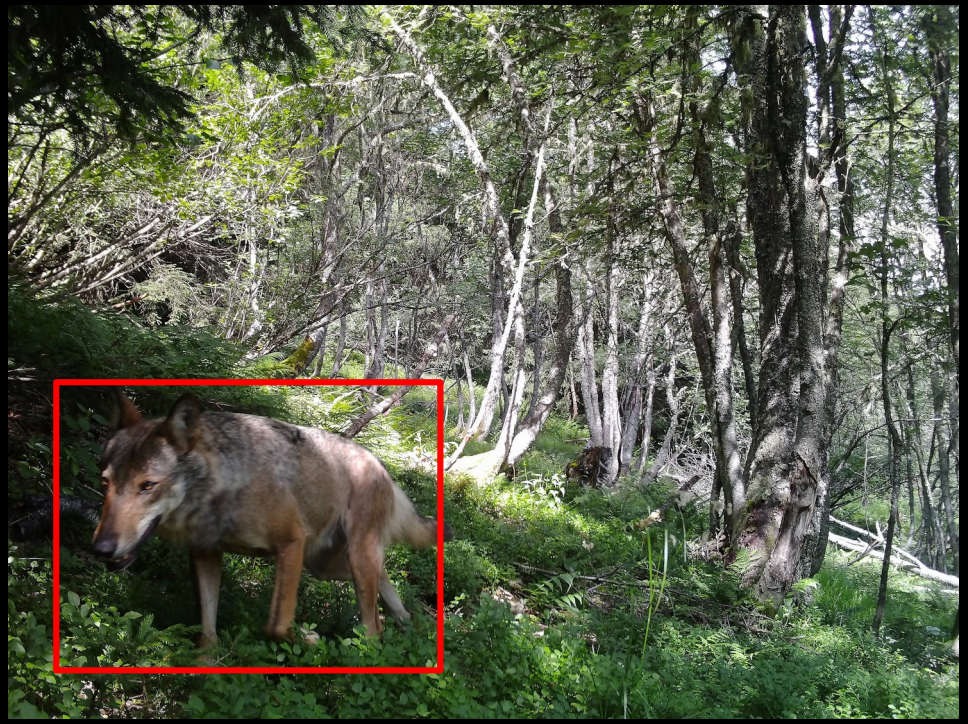
Les séquences de photos sont récupérées dans les cartes SD des caméras piège et déposées dans un pipeline pour leurs classification automatique

(Source: @obsorchamp tweeter)

# Orchamp vision

## MegaDetector
(Microsoft AI for Earth)

Est utilisé pour détecter la présence d'un animal sur la photo.
50 à 70% des images sont vides (déclenchement artefactuel lié au vent, etc)



(Source: @obsorchamp tweeter)

https://github.com/microsoft/CameraTraps/tree/main/detection

# Orchamp vision

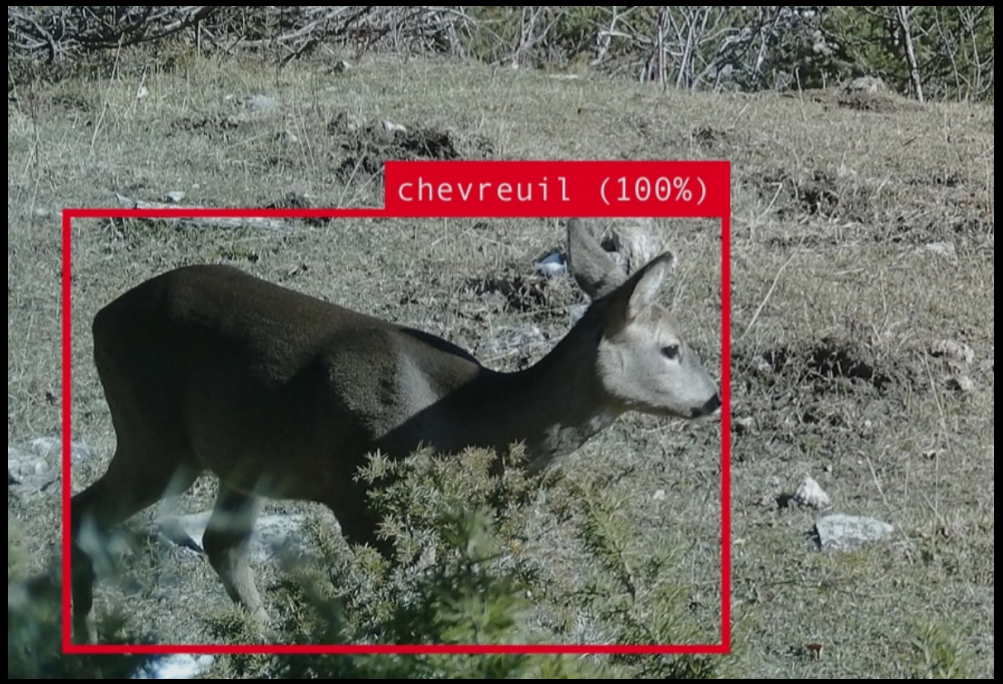
## DeepFaune

(projet collaboratif, porté entre autres par Vincent Miele)

https://www.deepfaune.cnrs.fr/

## Est utilisé pour classifier l'animal

DeepFaune est supporté par le PNRIA




Ref biblio:
https://www.biorxiv.org/content/10.1101/2022.03.15.484324v1

# Pipeline orchamp-vision

- Les données sont déposées dans un stockage iRods
  → plateforme Mantis de GRICAD

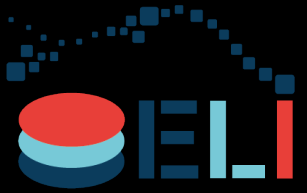
- Megadetector et DeepFaune tournent sur GPU
  → plateforme BigFoot de GRICAD

# Pipeline orchamp-vision


- Les données sont déposées dans un stockage iRods
  → plateforme Mantis de GRICAD

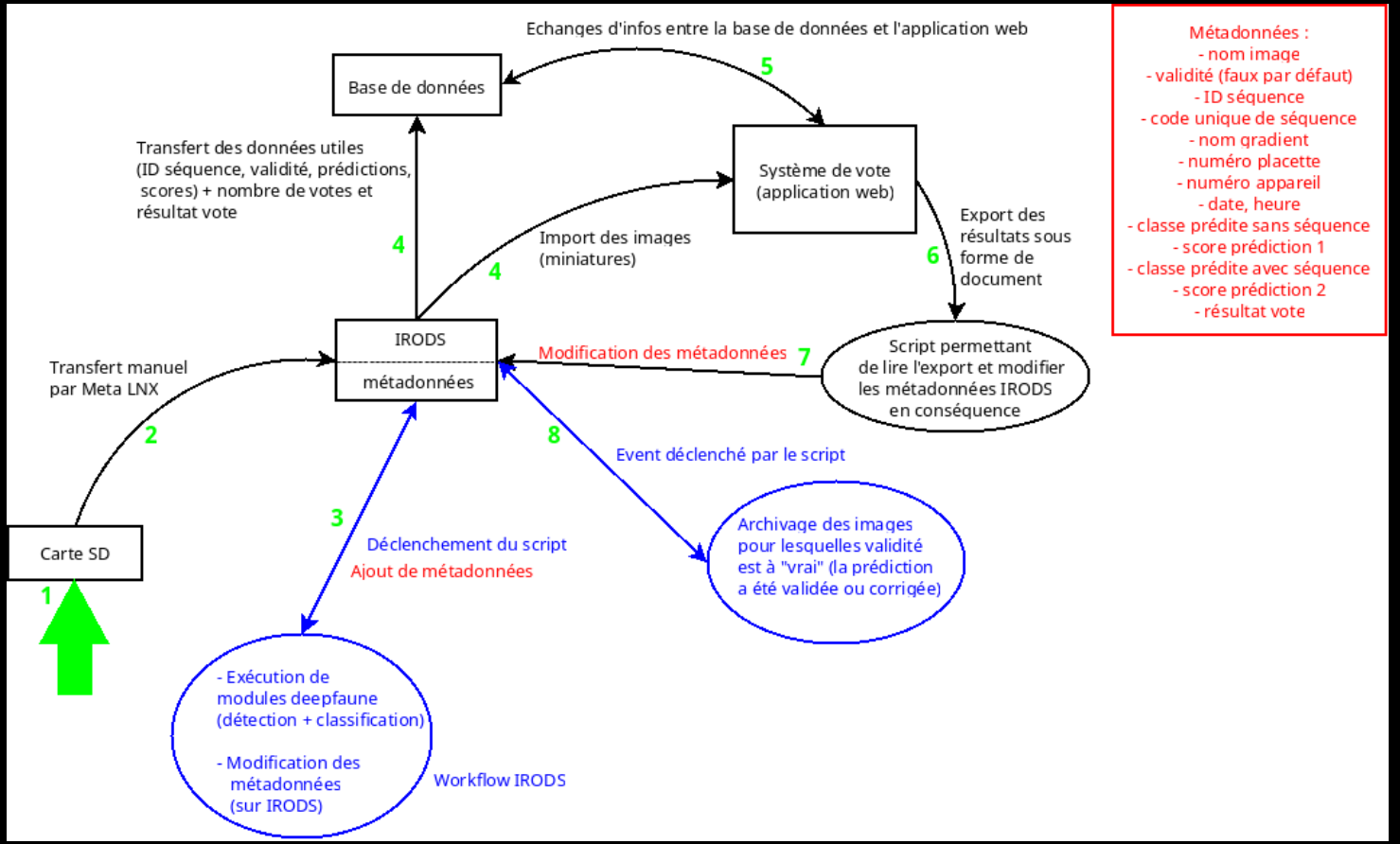

- Megadetector et DeepFaune tournent sur GPU
  → plateforme BigFoot de GRICAD

- Les données réduites seront synchronisées dans une base ElasticSearch, pour étude statistique
  → plateforme Eli de GRICAD




- Validation manuelle via interface web ad-hoc (Julien renaud)

# Pipeline orchamp-vision



# iRods, pourquoi



- iRods est un système de stockage distribué permettant de gérer de gros volumes de données dans un espace de nommage unique, géré “façon cloud”
- Il offre de multiples manières de l’interfacer avec d’autres systèmes (webDav, API python, API C, interface web,...)
- Système de meta-données avancé
- Système de règles

# Utilisation avancée d'iRods

- Importation d'images via MetaLNX (interface web iRods)
  - Importation automatique des données EXIF (date de prise de vue, coordonnées GPS, etc…) dans les meta-données iRods
  - Création automatique d'une version redimensionnée de l'image dans un sous-répertoire "thumbnails"
  - Déclenchement automatique d'un job de calcul sur le cluster BIGFOOT par dépot d'un fichier de contrôle dans la même interface

# Utilisation avancée d'iRods

- Importation d'images via MetaLNX (interface web iRods)
  - Importation automatique des données EXIF (date de prise de vue, coordonnées GPS, etc…) dans les meta-données iRods
  - Création automatique d'une version redimensionnée de l'image dans un sous-répertoire "thumbnails"
  - Déclenchement automatique d'un job de calcul sur le cluster BIGFOOT par dépot d'un fichier de contrôle dans la même interface
  - Synchronisation des meta-data avec Elasticsearch
- Mise à jour des meta-data par l'interface de validation

# Configuration iRods

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger


OAR est le gestionnaire de ressources permettant l'exploitation du cluster de GPU "Bigfoot"

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger
- Mise en place de la synchronisation des meta-data avec Elasticsearch

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger
- Mise en place de la synchronisation des meta-data avec Elasticsearch

# Plugin python

https://slides.com/jasoncoposky/cines-2020-rule-engine-plugins

```
# Installation du plugin sur tous les noeuds
# (clush est l'outil ClusterShell, permettant l'éxécution de commandes en // sur plusieurs
serveurs)
clush -bw @mantis "LANG=C apt-get install -y irods-rule-engine-plugin-python"
clush -bw @mantis "touch /etc/irods/core.py"
```

# Plugin python

https://slides.com/jasoncoposky/cines-2020-rule-engine-plugins

```
# Installation du plugin sur tous les noeuds
# (clush est l'outil ClusterShell, permettant l'éxécution de commandes en // sur plusieurs
serveurs)
clush -bw @mantis "LANG=C apt-get install -y irods-rule-engine-plugin-python"
clush -bw @mantis "touch /etc/irods/core.py"

# Editer `/etc/irods/server_config.json` sur tous les noeuds

[...]
"rule_engines": [
 {
 "instance_name" : "irods_rule_engine_plugin-python-instance",
 "plugin_name" : "irods_rule_engine_plugin-python",
 "plugin_specific_configuration" : {}
 },
 {
 "instance_name": "irods_rule_engine_plugin-irods_rule_language-instance",
[...]
```

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger
- Mise en place de la synchronisation des meta-data avec Elasticsearch

# Règle pour l’ajout des metadata

```
# Editer `/etc/irods/server_config.json` sur tous les noeuds
# Add `metadata` into `re_rulebase_set` (we can call it whatever we want)
# On every resource
[...]
"re_rulebase_set": [
 "metadata",
 "core"
 ],
[...]
```

# Règle pour l'ajout des metadata

```
# Editer `/etc/irods/server_config.json` sur tous les noeuds
# Add `metadata` into `re_rulebase_set` (we can call it whatever we want)
# On every resource
[...]
"re_rulebase_set": [
 "metadata",
 "core"
 ],
[...]

# Create the corresponding rule file (on every resource)

cat <<EOF >  /etc/irods/metadata.re
add_metadata_to_objpath(*str, *objpath, *objtype) {
 msiString2KeyValPair(*str, *kvp);
 msiAssociateKeyValuePairsToObj(*kvp, *objpath, *objtype);
}
getSessionVar(*name,*output) {
 *output = eval("str($"+++*name++")");
}
```

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger
- Mise en place de la synchronisation des meta-data avec Elasticsearch

# Install des dependances

```
# Là, encore, sur toutes les ressources de stockage irods:

clush -bw @mantis "apt-get install -y python-exif imagemagick python-pycurl"
```

Pour l'import auto des données exif dans les meta-data

Pour la création auto des thumbnails

Pour la soumission automatique de jobs via l'API restFull de OAR

# Configuration iRods

- Activation du plugin python du moteur de règles
- Créer une règle (dans le langage natif) permettant l'ajout de meta-données
- Installer les dépendances
- Créer les règles en python
  - Importation EXIF
  - Thumbnails
  - OAR job trigger
- Mise en place de la synchronisation des meta-data avec Elasticsearch

# Règles python


```
# Installation du fichier de règles sur toutes les ressources

clush -bw @mantis --copy core.py --dest /etc/irods/core.py
```

# Core.py : headers



```
#
# iRods python rules plugin inspired from multiple sources:
#
# https://indico.in2p3.fr/event/23075/contributions/90180/attachments/61890/84544/08_Moteur_Regles.pdf
# https://slides.com/jasoncoposky/cines-2020-rule-engine-plugins
# https://github.com/irods/irods_training/blob/main/beginner/irods_beginner_training_2019.pdf
# https://github.com/irods/irods_rule_engine_plugin_python

import pycurl
import json
import re
import exifread
import session_vars
import os
import subprocess
from genquery import *
from io import BytesIO

# Variables for the OAR jobs automatic submission
OAR_REST_API_PORT="6669"
OAR_REST_API_PATH="/oarapi-cigri/jobs"
OAR_CLIENT_CERT="/etc/cigri/ssl/cigri.crt"
OAR_CLIENT_KEY="/etc/cigri/ssl/cigri.key"
OAR_SCRIPT_PREFIX="./mantis/"

# Variables used by thumbnails auto generation
TMPDIR="/var/tmp/"
THUMBNAIL_SIZE="608x608"
DEFAULT_RESOURCE="imag"

# Resources that are excluded from rules that do things automatically
# (typically backup resources)
EXCLUDED_RESOURCES=['nigel-4.u-ga.fr','nigel-5.u-ga.fr']
```

# Core.py : functions for OAR

```
# Post a OAR job
def submit_oar_job(callback,user,cluster,script_name,path):
  headers = {'Accept': 'application/json','X-Remote-Ident': '{}'.format(user)}
  job = json.dumps({"scanscript": "", "command": "{}{} {}".format(OAR_SCRIPT_PREFIX,script_name,path)})
  callback.writeLine('serverLog', 'Submitting OAR job {} as {} on {}'.format(script_name,user,cluster))
  c = pycurl.Curl()
  buffer = BytesIO()
  c.setopt(pycurl.URL, "https://{}:{}{}".format(cluster,OAR_REST_API_PORT,OAR_REST_API_PATH))
  c.setopt(pycurl.HTTPHEADER, ['Content-Type:application/json','Accept:application/json','X-Remote-Ident:{}'.format(user)])
  c.setopt(pycurl.POST, 1)
  c.setopt(pycurl.POSTFIELDS, job)
  c.setopt(pycurl.SSL_VERIFYPEER, 0)
  c.setopt(pycurl.SSL_VERIFYHOST, 0)
  c.setopt(pycurl.SSLCERT, OAR_CLIENT_CERT)
  c.setopt(pycurl.SSLKEY, OAR_CLIENT_KEY)
  c.setopt(c.WRITEFUNCTION, buffer.write)
  #c.setopt(c.VERBOSE, True)
  c.perform()
  if c.getinfo(pycurl.RESPONSE_CODE) != 201:
    callback.writeLine('serverLog', 'OAR job submit failed')
  body = buffer.getvalue()
  callback.writeLine('serverLog', body)
  c.close()

# Get the username of the client
def get_username(rule_args, callback, rei):
  username = ''
  var_map = session_vars.get_map(rei)
  userrec = var_map.get('client_user','')
  if userrec:
    username = userrec.get('user_name','')
  return username
```

# Core.py : functions for EXIF

```
# Function to extract exif of an image and update metadata of the file
def exif_python_rule(rule_args, callback, rei):
  file_path = str(rule_args[0])
  obj_path = str(rule_args[1])
  exiflist = []
  with open(file_path, 'rb') as f:
    tags = exifread.process_file(f, details=False)
    for (k, v) in tags.iteritems():
      if k not in ('JPEGThumbnail', 'TIFFThumbnail', 'Filename', 'EXIF MakerNote'):
        exifpair = '{0}={1}'.format(k, v)
        exiflist.append(exifpair)
  exifstring = '%'.join(exiflist)
  #callback.writeLine('serverLog', 'Exifstring={}'.format(exifstring))
  callback.add_metadata_to_objpath(exifstring, obj_path, '-d')
  callback.writeLine('serverLog', 'PYTHON EXIF RULE complete')
```

# Core.py : functions for thumbnails

```
# Check if there's a "thumbnails" sub-directory into the given path
def thumbnails_exists(callback, path):
  found=False
  collection=os.path.dirname(path)
  search=collection+"/thumbnails"
  for coll in Query(callback, columns="COLL_NAME", conditions="COLL_NAME = '{}'".format(search)):
    found = True
  return found

# Function to generate a thumbnail
def generate_thumbnail(rule_args, callback, rei):
  file_path = str(rule_args[0])
  obj_path = str(rule_args[1])
  callback.writeLine('serverLog', 'Generating Thumbnail for {}'.format(file_path))
  result=subprocess.check_output(['convert','-resize',THUMBNAIL_SIZE,file_path,TMPDIR+os.path.basename(file_path)])
  if result != "":
    callback.writeLine('serverLog', result)
  collection=os.path.dirname(obj_path)+"/thumbnails"
  username=get_username(rule_args,callback,rei)
  os.environ['clientUserName'] = username
  result=subprocess.check_output(['iput','-f','-R',DEFAULT_RESOURCE,TMPDIR+os.path.basename(file_path),collection])
  if result != "":
    callback.writeLine('serverLog', result)
  os.unlink(TMPDIR+os.path.basename(file_path))
```

# Core.py : misc functions



```
# Do a query to get the server hostname of the resource
# (used to filter resources that do not do computations)
# check EXCLUDES_RESOURCES config variable
def get_resource_location(callback,resource):
  for loc in Query(callback,"RESC_LOC",conditions = "RESC_NAME = '{}'".format(resource)):
        pass
  return loc
```

# Core.py : acPostProcForPut - Partie 1

```
# Rule executed after a Put
def acPostProcForPut(rule_args, callback, rei):
  sv = session_vars.get_map(rei)
  phypath = sv['data_object']['file_path']
  objpath = sv['data_object']['object_path']
  resource = sv['data_object']['resource_name'] # More keys can be found into plugins/api/src/get_file_descriptor_info.cpp

  # EXIF processing rule
  # If the file ends with .jpg, the EXIF data is extracted
  # and imported into the metadata of the file
  if phypath[-4:] == '.jpg' or phypath[-4:] == '.JPG':
    callback.writeLine('serverLog', 'Exec EXIF Python Rule')
    remote_rule = "exif_python_rule('%s', '%s')" % \
                  (phypath, objpath)
    location = get_resource_location(callback,resource)
    # Exclude backup resources here (no processing on data for thoses resources)
    exclude = False
    for res in EXCLUDED_RESOURCES:
      if location == res:
        exclude = True
    if exclude :
      callback.writeLine('serverLog', 'No EXIF processing for {}'.format(location))
    # Send the processing on the server hosting the data
    else:
      callback.writeLine('serverLog', 'Processing EXIF rule on {}'.format(location))
      callback.remoteExec(location, '', remote_rule, '')
```

# Core.py : acPostProcForPut - Partie 2

```
# OAR job submission
# If a file is created with a name of the form:
#    <scriptname>.<hostname>.oar-autosubmit
# Then, the OAR script OAR_SCRIPT_PREFIX/<scriptname> is submitted to the
# <hostname> OAR frontend restful API.
if phypath[-15:] == '_oar-autosubmit':
  params=phypath.rsplit('/',1)[1].split('_')
  if len(params) == 3:
    script_name=params[0]
    cluster=params[1]
    username=get_username(rule_args,callback,rei)
    submit_oar_job(callback,username,cluster,script_name,objpath)

# Thumbnails generation
if phypath[-4:] == '.jpg' or phypath[-4:] == '.JPG':
  if thumbnails_exists(callback,objpath):
    location = get_resource_location(callback,resource)
    remote_rule = "generate_thumbnail('%s', '%s')" % \
                  (phypath, objpath)
    # Exclude backup resources here (no processing on data for thoses resources)
    for res in EXCLUDED_RESOURCES:
      if location == res:
        exclude = True
    if exclude :
      callback.writeLine('serverLog', 'No THUMBNAIL processing for {}'.format(location))
    # Send the processing on the server hosting the data
    else:
      callback.remoteExec(location, '', remote_rule, '')

callback.writeLine('serverLog', 'PYTHON - acPostProcForPut() complete')
```

# Utilisation

# Utilisation: import des images

- Les EXIFS sont automatiquements importés à partir du moment où le fichier porte une extension .jpg ou .JPG
- Les thumbnails sont créés si une sous-collection iRods “thumbnails” existe

# Utilisation: lancement du job de classification

Le lancement du job est déclenché par l'upload d'un fichier spécial comportant les informations essentielles dans son nom:

**`orchamp.oar_bigfoot.u-ga.fr_oar-autosubmit`**

- `orchamp.oar` : Nom du script lancé par le job (doit se trouver sur le cluster dans le répertoire `~/mantis/`
- `bigfoot.u-ga.fr` : Nom d'hôte de l'API OAR du cluster à utiliser
- `oar-autosubmit` : Extension trigger

Le job envoie le script avec le chemin complet du fichier en argument, ex:

```
./mantis/orchamp.oar /mantis/home/bzizou/orchamp.oar_bigfoot.u-ga.fr_oar-autosubmit
```

(cela permet éventuellement de passer des informations directement à l'intérieur de ce fichier, mais surtout de savoir dans quelle collection iRods on travaille)

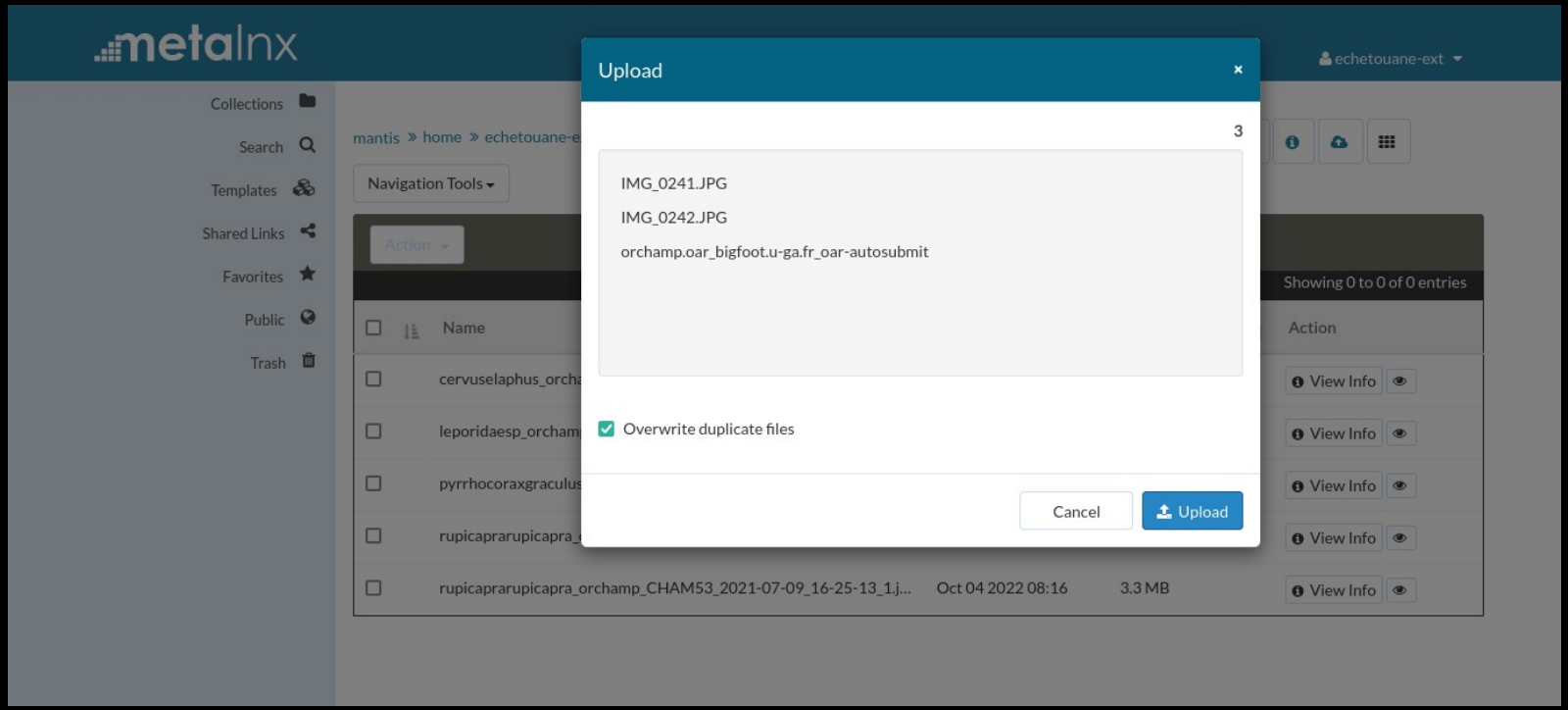
# Utilisation: exemple de job

```
$ cat ~/mantis/orchamp.oar
#!/bin/bash

#OAR -l /nodes=1/gpu=1,walltime=10:00:00
#OAR -p gpumodel='V100'
#OAR --project orchampvision

source /applis/site/nix.sh

irm -f $1 # removing the file that triggered the script so
          # that there is only images left in the collection

# moving to the right directory to use tensorflow with nix
cd /bettik/PROJECTS/pr-orchampvision/COMMON/pytorch-nixenv
# starting the nix-shell that executes the python script
nix-shell --command "python pipeline.py $(dirname $1)"
```

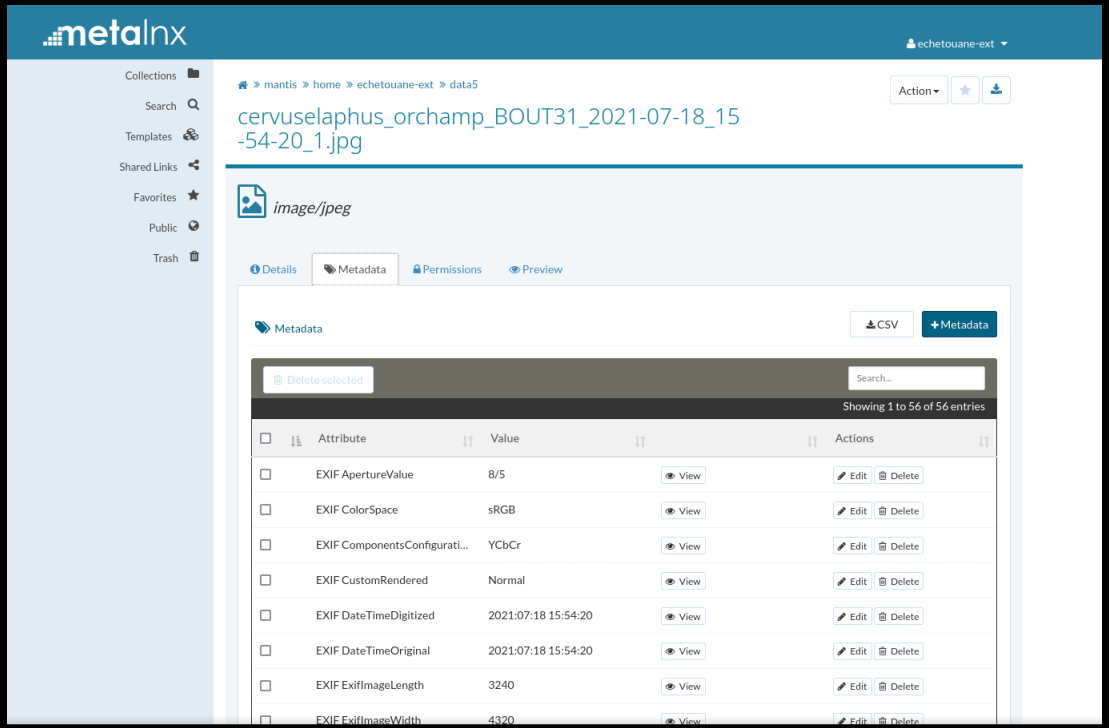
# Utilisation: log iRods d'une soumission OAR

```
root@nigel-0:~# tail  -18 /var/lib/irods/log/rodsLog.2022.04.11


Apr 15 18:46:10 pid:25895 NOTICE: writeLine: inString = PYTHON - acPostProcForPut() complete
Apr 15 18:51:02 pid:26123 NOTICE: writeLine: inString = Submitting OAR job orchamp.oar as
bzizou on bigfoot.u-ga.fr
Apr 15 18:51:02 pid:26123 NOTICE: writeLine: inString = {
   "id" : 8538,
   "cmd_output" : "[ADMISSION RULE] Modify resource description with type constraints\
nOAR_JOB_ID=8538\n",
   "api_timestamp" : 1650041462,
   "links" : [
      {
         "href" : "/oarapi-cigri/jobs/8538",
         "rel" : "self"
      }
   ]
}


Apr 15 18:51:02 pid:26123 NOTICE: writeLine: inString = PYTHON - acPostProcForPut() complete
```

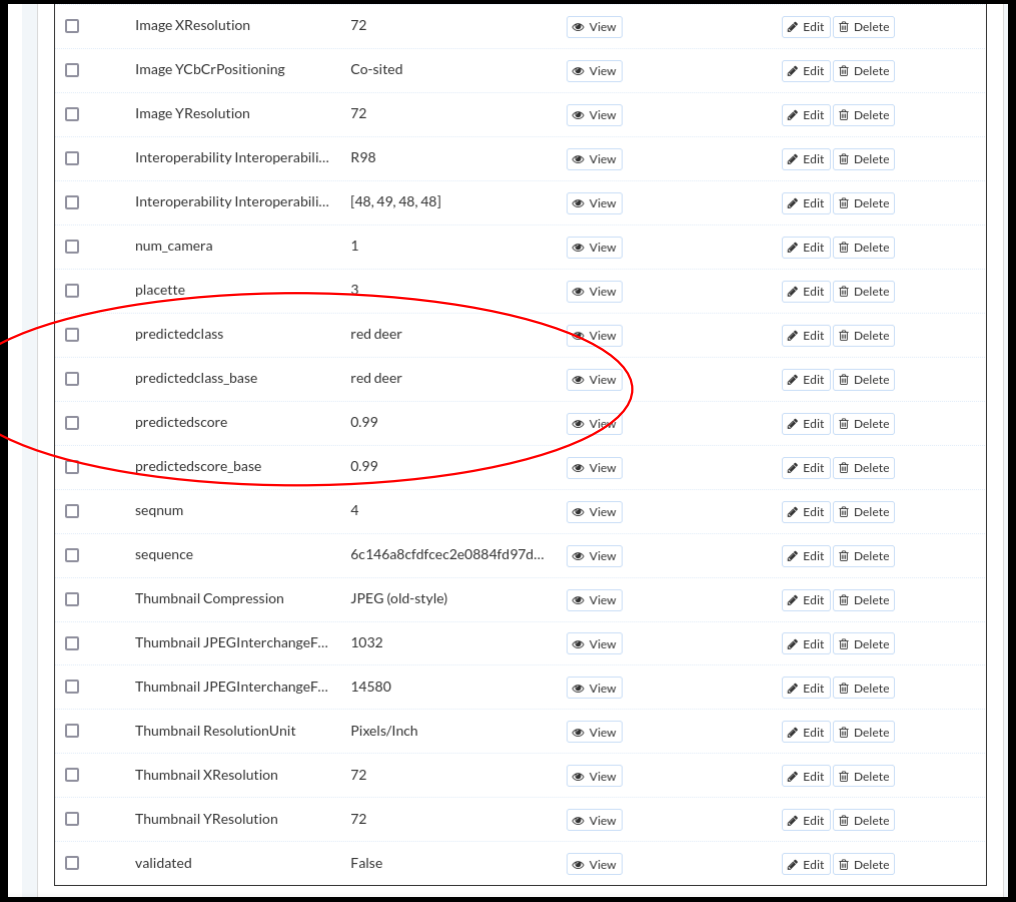
# Exemple d’importation



# Exemple: metadonnées



# Exemple: classification effectuée



| | | | | |
|---|---|---|---|---|
| ☐ | Image XResolution | 72 | View | Edit Delete |
| ☐ | Image YCbCrPositioning | Co-sited | View | Edit Delete |
| ☐ | Image YResolution | 72 | View | Edit Delete |
| ☐ | Interoperability Interoperabili... | R98 | View | Edit Delete |
| ☐ | Interoperability Interoperabili... | [48, 49, 48, 48] | View | Edit Delete |
| ☐ | num_camera | 1 | View | Edit Delete |
| ☐ | placette | 3 | View | Edit Delete |
| ☐ | predictedclass | red deer | View | Edit Delete |
| ☐ | predictedclass_base | red deer | View | Edit Delete |
| ☐ | predictedscore | 0.99 | View | Edit Delete |
| ☐ | predictedscore_base | 0.99 | View | Edit Delete |
| ☐ | seqnum | 4 | View | Edit Delete |
| ☐ | sequence | 6c146a8cfdfcec2e0884fd97d... | View | Edit Delete |
| ☐ | Thumbnail Compression | JPEG (old-style) | View | Edit Delete |
| ☐ | Thumbnail JPEGInterchangeF... | 1032 | View | Edit Delete |
| ☐ | Thumbnail JPEGInterchangeF... | 14580 | View | Edit Delete |
| ☐ | Thumbnail ResolutionUnit | Pixels/Inch | View | Edit Delete |
| ☐ | Thumbnail XResolution | 72 | View | Edit Delete |
| ☐ | Thumbnail YResolution | 72 | View | Edit Delete |
| ☐ | validated | False | View | Edit Delete |

# Conclusion

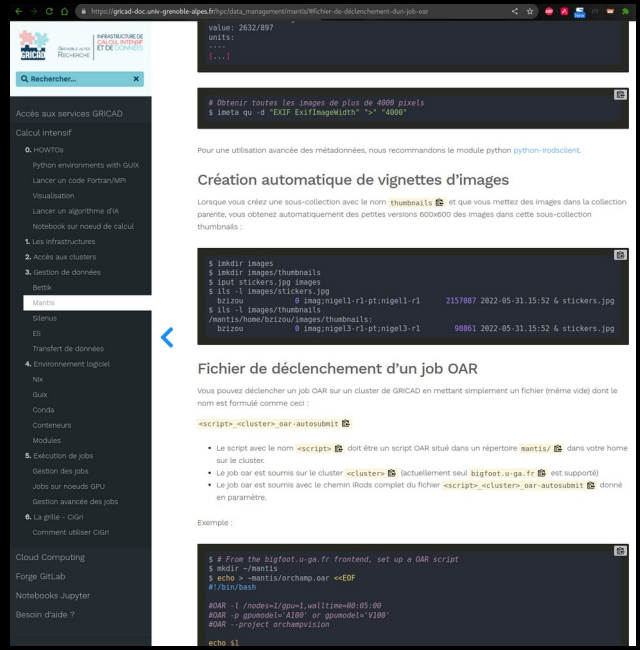
# Perspectives

- Il reste encore du travail:
  - Synchro des données dans ElasticSearch (ELI)
  - Réalisation de l’interface web de validation
  - Archivage

- Travail similaire à faire plus tard sur ORCHAMPsound, avec les fichiers d’enregistrement audio

# Conclusions

- Travail très collaboratif: GRICAD, LECA, LBBE, DeepFaune
- Un bel exemple de cas d'usage concrêt qui amène à des mises en oeuvre techniques qui peuvent être utiles à tous
- Preuve de l'intérêt des collaborations Mesocentre/Laboratoire pour aller plus loin dans une bonne exploitation des différentes technologies à notre disposition, au service de la recherche
- De belles perspectives, il reste du travail (elastic, ORCHAMPsound)




Extrait de la documentation de GRICAD
Mettant à la dispo de tous les developpements
Effectués initialement pour le projet ORCHAMP

# Merci!

https://orchamp.osug.fr/

https://gricad.univ-grenoble-alpes.fr/

https://www.biorxiv.org/content/10.1101/2022.03.15.484324v1

https://www.deepfaune.cnrs.fr/

https://irods.org/

Des questions?