INNUENDO Platform Documentation

Release 1

Bruno Ribeiro-Gonçalves

Feb 14, 2019

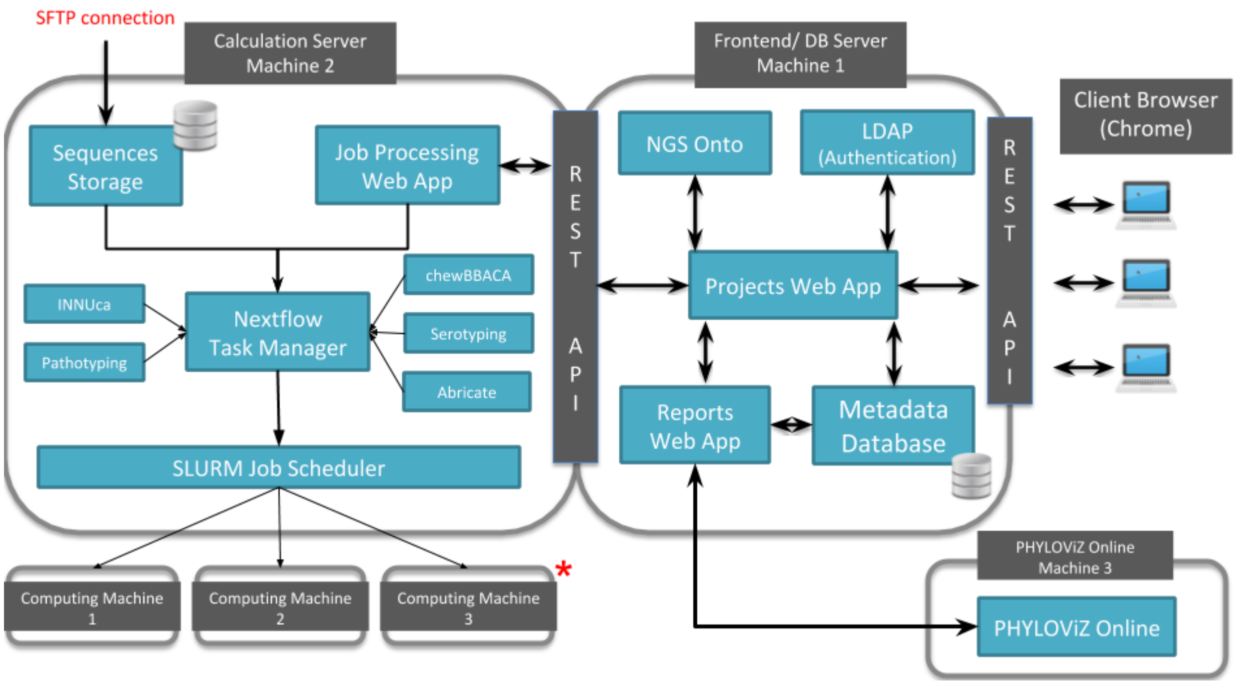
Dependencies

A novel cross-sectorial platform for the integration of genomics in surveillance of foodborne pathogens

Multinational outbreaks of foodborne pathogens cause considerable threats to European public health. Implementing whole genome sequencing (WGS) in routine surveillance and outbreak investigations is becoming a strategic goal for many public health authorities all over the world. With this in mind we developed the initiative INNUENDO, which aims to deliver a cross-sectorial framework for the integration of bacterial WGS in routine surveillance and epidemiologic investigations.

INNUENDO platform is divided into two distinct applications that communicate between each other. The first one, the **INNUENDO frontend server**, comprises the user web interface and mechanisms to allow secure user authentication with LDAP and data storage into a dedicated database. It also communicates with the **INNUENDO process controller**, which was developed with the aim of working as a bridge to allow running analytical procedures on a laptop or in a High Performance Computer (HPC), with the help of SLURM process manager and Nextflow.

There is also a **docker-compose** version of the platform that can be easily installed with a few commands.



CHAPTER 1

Contents

The documentation of the **INNUENDO Platform** follows the below structure:

- *Dependencies*
- *Installation*
- *Docker-Compose*
- *Usage*
- *Admins: Troubleshooting and Backup*

1.1 Dependencies List

The INNUENDO Platform is composed of a set of modules that communicate between each other by RESTful APIs. However, there are also other dependencies that are required so that the servers can run as expected.

All those components described bellow are necessary for a multi-machine and individual component installation. You can also install all the application using this approach or by using the Docker-Compose module developed for this purpose.

1.1.1 Main modules and their dependencies

*Described on parent page

- **Frontend Server**
  - Nginx
  - NodeJS*
  - Bower*
  - Allegrograph client*
- **Process Controller Server**

    - Nginx
    - Nextflow
    - FlowCraft
    - Allegrograph client
- **Reports Application**
    - Nginx
    - NodeJS*
    - Bower*
- **SLURM**
    - MariaDB*
    - Munge*
- **LDAP**
    - LDAP server*
    - phpldapadmin*
    - LDAP client (third party authentication)*

1.2 Nginx

Nginx is a web-server used to allow communication between different machines and expose the Frontend application to the web if required.

Each Application has a RESTful API used for the communication. The route for each of these applications needs to be mapped into the nginx configuration file for each independent machine.

1.2.1 Installation

Install the Nginx software from the package manager.

```
sudo apt-get install nginx
```

1.2.2 Create a new configuration file

Add a new configuration file named innuendo.com which will be used to allow Nginx to be set as a reverse proxy for the AllegroGraph, INNUENDO_REST_API application and Reports application.

Fill with the following.

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        listen 443 ssl;
        server_name _;

```

(continues on next page)

(continued from previous page)

```
        ssl_certificate /etc/nginx/ssl/nginx.crt;
        ssl_certificate_key /etc/nginx/ssl/nginx.key;

        location /app {
            proxy_pass http://localhost:5000;
        }

        location / {
            proxy_pass http://localhost:10035;
        }

    # Use this location if the INNUENDO_PROCESS_CONTROLLER is on the same
    # machine as the INNUENDO_REST_API. Otherwise, comment this route.
        location /jobs {
        proxy_pass http://localhost:5001;
    }

        location /ldap/ {
        rewrite ^/ldap/(.*) /$1  break;
        proxy_pass http://localhost:81;
    }

    location /reportsApp/ {
        rewrite ^/reportsApp/(.*) /$1  break;
        proxy_pass http://localhost:82;
    }

}
```

For the INNUENDO Reports application, create a reports.com file and add the following.

```
server {
    listen       82;
    server_name  localhost;

    #charset koi8-r;

    #access_log  logs/host.access.log  main;
    root   /usr/src/app;
    index  index.html index.htm;

    location / {
        try_files $uri /index.html;
    }
}
```

If the INNUENDO_PROCESS_CONTROLLER is on a different machine, create also a innuendo.com file and add the following.

```
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        listen 443 ssl;
        server_name _;

```

(continues on next page)

(continued from previous page)

```
        ssl_certificate /etc/nginx/ssl/nginx.crt;
        ssl_certificate_key /etc/nginx/ssl/nginx.key;

        location /jobs {
        #rewrite ^/jobs/(.*) /$1  break;
        proxy_pass http://localhost:5001;
    }
}
```

1.2.3 Create a SSL certificate

If a encrypted connection is required, you will need to generate an SSL certificate. Do that in all the independent machines that require an encrypted connection, such as the machine with the INNUENDO_REST_API. Do that with the following commands.

```
sudo mkdir /etc/nginx/ssl
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/nginx/ssl/nginx.
↪key -out /etc/nginx/ssl/nginx.crt
```

1.2.4 Add to sites-available

For the configuration files be used by Nginx, they need to be located into the sites-available folder. You can do that with the following commands.

```
# Move the configuration file to the sites-available folder of Nginx
mv innuendo.com /etc/nginx/sites-available/

# Move the reports configuration file to the sites-available folder of Nginx
mv reports.com /etc/nginx/sites-available/

# Enter the sites-available folder
cd /etc/nginx/sites-available/

# Link the innuendo.com file to one in the sites-enabled folder
ln -s /etc/nginx/sites-available/innuendo.com /etc/nginx/sites-enabled/

# Link the reports.com file to one in the sites-enabled folder
ln -s /etc/nginx/sites-available/innuendo.com /etc/nginx/sites-enabled/
```

1.2.5 Restart Nginx

Restart Nginx so that the changes can take place.

```
sudo service restart nginx
```

1.3 Allegrograph

Allegrograph is a triplestore database used in the INNUENDO Platform to store relationships between everything, from strains in projects to the processes that are run on those strains in a specific project.

Currently it uses an unpaid version with store to about 1 million triples. If required, a paid version can be obtained to obtain more storage.

1.3.1 Installation

Install some general dependencies.

```
sudo apt-get update
sudo apt-get install -y git python-pip libpq-dev libcurl4-openssl-dev python-dev␣
↪libsasl2-dev libldap2-dev libssl-dev wget
```

Get Allegrograph server installer from the INNUENDO releases.

```
# Create a directory to store the files
mkdir allegrograph

# Enter the directory
cd allegrograph

# Download the server files
wget https://github.com/bfrgoncalves/INNUENDO_files/releases/download/1.0.0/agraph-6.
↪0.2-linuxamd64.64.tar.gz
```

Uncompress the downloaded files.

```
tar zxf agraph-6.0.2-linuxamd64.64.tar.gz
```

Install the Allegrograph server in an non-interactive way. You can change the file locations and username by changing the inputs in the directives.

```
agraph-6.0.2/install-agraph ./agraph --non-interactive \
    --config-file "./agraph/lib/agraph.cfg" \
    --data-dir "./agraph/data" \
    --log-dir "./agraph/log" \
    --pid-file "./agraph/data/agraph.pid" \
    --runas-user "innuendo" \
    --create-runas-user \
    --port 10035 \
    --super-user "innuendo" \
    --super-password "innuendo_allegro"
```

Launch the allegrograph server. It needs to be running for the Frontend server and the Controller to work.

```
./agraph/bin/agraph-control --config /Allegrograph/agraph/lib/agraph.cfg start
```

1.4 PostgreSQL

PostgreSQL is the default database used in the INNUENDO Platform for data storage. It needs to be installed in the same machine as the Frontend server or configured in such a way that the Frontend server can access to it.

1.4.1 Installation

```
sudo apt-get update
sudo apt-get install postgresql postgresql-contrib
```

1.4.2 Create Postgres User

Enter with the default "postgres" user and create a new user to be used in the Platform. Change the version according to the installed postgres version. Is recommended to use postgres version < 10.

```
sudo -u postgres /usr/lib/postgresql/9.X/bin/createuser innuendo
```

1.4.3 Create the Database

Launch psql with the default postgres user.

```
sudo -u postgres psql postgres
```

Inside psql, set a password for the default postgres user.

```
postgres=# \password postgres
```

Change the permissions of the previously created user to allow the creation of databases.

```
postgres=# ALTER USER innuendo CREATEDB;
```

Create the innuendo database.

```
postgres=# CREATE DATABASE innuendo OWNER innuendo;
```

Exit psql.

```
postgres=# \q
```

1.4.4 Change Configuration file

Locate the postgreSQL pg_hba.conf file. It has all the information regarding access security to the database. It is required to change some of the parameters.

The file should be at */etc/postgresql/9.X/main/*

Open it and replace all the METHOD column to *trust*

Restart postgreSQL.

```
sudo service postgresql restart
```

1.4.5 Set password for the INNUENDO user

Launch psql with the created user.

```
sudo -u innuendo psql innuendo
```

Inside psql, set a password for the innuendo user.

```
postgres=# \password innuendo
```

Exit psql.

```
postgres=# \q
```

1.4.6 Change Configuration file (AGAIN)

Open the *pg_hba.conf* file and replace all METHOD column to *md5*.

Restart postgreSQL.

```
sudo service postgresql restart
```

1.4.7 Create/Load the database structure

Now you can load the database structure using a set of commands defined by the Flask-Migrate package. It should be available after installing the Frontend server and all its dependencies.

Inside de **INNUENDO_REST_API** folder run.

```
# Initialize the database and build a migrations directory
./manage.py db init --multidb

# Sets the new version of the database
./manage.py db migrate

# Recreates the database with the newest version
./manage.py db upgrade
```

1.5 LDAP

LDAP is a centralized authentication system that allow users to authenticate in multiple applications only using a single account. It requires the installation of a server application in the service provider machine and clients in all the machines that want to authenticate.

Before installing LDAP, define an LDAP domain that will be used for the server creation and for client authentication.

1.5.1 Install LDAP Server

To install the LDAP server, run the following command.

```
sudo apt-get install slapd ldap-utils

# Choose these options on the installer
Omit openLDAP config: No
base DN of the LDAP directory: innuendo.com
```

(continues on next page)

(continued from previous page)

```
organization name: innuendo
Database backend to use: HDB
Database removed when slapd is purged: No
MOve old database: Yes
Allow LDAPv2 protocol: No
```

For an easier integration with LDAP and monitoring, it is advised to install phpldapadmin, an application that provides a web-interface to deal with LDAP without using the command line. To install, run the following.

```
sudo apt-get install phpldapadmin
```

You can follow the instructions on this tutorial for an easier configuration. https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-openldap-and-phpldapadmin-on-an-ubuntu-14-04-server

In phpldapadmin, do the following steps:

- **Create two Organizational Units.**
  - groups
  - users
- **Add two Posix Groups to the groups entry created.**
  - admin
  - innuendo_users
- **Add Generic User Accounts.**
  - Add email to the account
  - Add it to the admin or innuendo_users

1.5.2 Install LDAP Client

To install the LDAP client needed to authenticate to the server, follow the tutorial in the link bellow.

https://www.digitalocean.com/community/tutorials/how-to-authenticate-client-computers-using-ldap-on-an-ubuntu-12-04-vps

1.5.3 Change new User Skel structure

Is necessary to change the skel of user creation so that some folders are created upon user definition. They are required to store the fastq files and files belonging to job submission.

Go to the skel folder and do the following.

```
# Enter skel folder
cd /etc/skel
# Create ftp and jobs folder
sudo mkdir ftp jobs
# Add files folder
sudo mkdir ftp/files
```

After completing these steps, two files are required to change the permissions when creating the folders for the users.

Create a file named change_ldap_user_permissions_innuendo.sh and add the following.

```
#!/bin/sh
chown root:root /mnt/innuendo_storage/users/$PAM_USER
chown root:root /mnt/innuendo_storage/users/$PAM_USER/ftp

chown ubuntu:ubuntu /mnt/innuendo_storage/users/$PAM_USER/jobs
```

This supposes an innuendo_storage folder inside the /mnt folder and a user running the application called ubuntu. To know more about how to mount folders between machines check the Configure NFS section.

After creating the permissions file, add it to the pam common-session file at /etc/pam.d/common-session to trigger the file permissions substitution.

```
# /etc/pam.d/common-session - session-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define tasks to be performed
# at the start and end of sessions of *any* kind (both interactive and
# non-interactive).
#
# As of pam 1.0.1-6, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
session [default=1]                     pam_permit.so
# here's the fallback if no module succeeds
session requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
session required                        pam_permit.so
# The pam_umask module will set the umask according to the system default in
# /etc/login.defs and user settings, solving the problem of different
# umask settings with different shells, display managers, remote sessions etc.
# See "man pam_umask".
session optional                        pam_umask.so
# and here are more per-package modules (the "Additional" block)
session required        pam_unix.so
session optional                        pam_ldap.so
session optional        pam_systemd.so
# end of pam-auth-update config

session required        pam_mkhomedir.so skel=/etc/skel umask=0022
session optional        pam_exec.so /usr/local/bin/change_ldap_user_permissions_
↪innuendo.sh
```

After replacing the required lines in the files, run the following command to restart the ldap client service.

```
sudo /etc/init.d/nscd restart
```

1.5.4 Setup SFTP (SSH) with LDAP

For Secure File Transfer, we will use the properties of SSH to allow the file tranfer. For that, we need to change the properties of the SSH configuration file.

Open the file with the following.

```
sudo nano /etc/ssh/sshd_config
```

At the end of the file, replace the Subsystem line and add the two Match Group entries described bellow. This will only allow SFTP connection of the innuendo users and will only allow to access to their home directory.

```
#Subsystem sftp /usr/lib/openssh/sftp-server
Subsystem sftp internal-sftp

# Set this to 'yes' to enable PAM authentication, account processing,
# and session processing. If this is enabled, PAM authentication will
# be allowed through the ChallengeResponseAuthentication and
# PasswordAuthentication.  Depending on your PAM configuration,
# PAM authentication via ChallengeResponseAuthentication may bypass
# the setting of "PermitRootLogin without-password".
# If you just want the PAM account and session checks to run without
# PAM authentication, then enable this but set PasswordAuthentication
# and ChallengeResponseAuthentication to 'no'.
UsePAM yes

Match Group innuendo-users
    ChrootDirectory %h/ftp
    AllowTCPForwarding no
    X11Forwarding no
    ForceCommand internal-sftp

Match Group admin
    ChrootDirectory %h/ftp
    AllowTCPForwarding no
    X11Forwarding no
    ForceCommand internal-sftp
```

After replacing the required lines in the file, restart SSH.

```
sudo /etc/init.d/ssh restart
```

1.6 SLURM

SLURM is a cluster management and job scheduling system that is used in the INNUENDO Platform to control job submission and resources between machines or in individual machines.

It requires a Master node, which will control all other nodes, and Slaves, which will run the jobs controlled by the master.

1.6.1 Installation

SLURM requires a set of software dependencies to work. We will need to install MariaDB (Only on the Master) for the SLURM Accounting module and also Munge for the communication between each machine (On each machine).

```
sudo apt-get install mariadb-server mariadb-devel
sudo apt-get install munge munge-libs munge-devel
```

Starting with Munge, first need to create a secret key on the Server for the communication between machines. First, we install rng-tools to properly create the key.

```
sudo apt-get install rng-tools
rngd -r /dev/urandom
```

Now, we create the secret key. You only have to do the creation of the secret key on the server.

```
/usr/sbin/create-munge-key -r

# Create key and change permissions and ownership
dd if=/dev/urandom bs=1 count=1024 > /etc/munge/munge.key
chown munge: /etc/munge/munge.key
chmod 400 /etc/munge/munge.key
```

After the secret key is created, you will need to send this key to all of the compute nodes.

```
# Example sending the key to a slave node called compute-1. You might
need to change the name with the machine domain
scp /etc/munge/munge.key root@compute-1:/etc/munge
```

Now, we SSH into every node and correct the permissions as well as start the Munge service.

```
# Change key permissions
chown -R munge: /etc/munge/ /var/log/munge/
chmod 0700 /etc/munge/ /var/log/munge/

# Start Munge service on the computing nodes
systemctl enable munge
systemctl start munge
```

To test Munge, you can try to access another node with Munge from your master node.

```
# Example access to node compute-1
munge -n
munge -n | unmunge
munge -n | ssh compute-1 unmunge
remunge
```

After all other dependencies are installed, you can now install SLURM with the following command.

```
sudo apt-get install slurm-llnl
```

1.6.2 SLURM Configuration

For SLURM configuration, we need to create a *slurm.conf* file and distribute it between all machines. We also need to define the *slurmdbd.conf* for the SLURM accouting.

Example *slurm.conf*

```
# slurm.conf
#
# See the slurm.conf man page for more information.
#
ClusterName=linux
ControlMachine=slurmctld
ControlAddr=slurmctld
#BackupController=
#BackupAddr=
```

(continues on next page)

(continued from previous page)

```
#
SlurmUser=slurm
#SlurmdUser=root
SlurmctldPort=6817
SlurmdPort=6818
AuthType=auth/munge
#JobCredentialPrivateKey=
#JobCredentialPublicCertificate=
StateSaveLocation=/var/lib/slurmd
SlurmdSpoolDir=/var/spool/slurmd
SwitchType=switch/none
MpiDefault=none
SlurmctldPidFile=/var/run/slurmd/slurmctld.pid
SlurmdPidFile=/var/run/slurmd/slurmd.pid
ProctrackType=proctrack/linuxproc
#PluginDir=
CacheGroups=0
#FirstJobId=
ReturnToService=0
#MaxJobCount=
#PlugStackConfig=
#PropagatePrioProcess=
#PropagateResourceLimits=
#PropagateResourceLimitsExcept=
#Prolog=
#Epilog=
#SrunProlog=
#SrunEpilog=
#TaskProlog=
#TaskEpilog=
#TaskPlugin=
#TrackWCKey=no
#TreeWidth=50
#TmpFS=
#UsePAM=
#
# TIMERS
SlurmctldTimeout=300
SlurmdTimeout=300
InactiveLimit=0
MinJobAge=300
KillWait=30
Waittime=0
#
# SCHEDULING
SchedulerType=sched/backfill
#SchedulerAuth=
#SchedulerPort=
#SchedulerRootFilter=
SelectType=select/cons_res
SelectTypeParameters=CR_CPU_Memory
FastSchedule=1
#PriorityType=priority/multifactor
#PriorityDecayHalfLife=14-0
#PriorityUsageResetPeriod=14-0
#PriorityWeightFairshare=100000
#PriorityWeightAge=1000
```

(continues on next page)

(continued from previous page)

```
#PriorityWeightPartition=10000
#PriorityWeightJobSize=1000
#PriorityMaxAge=1-0
#
# LOGGING
SlurmctldDebug=3
SlurmctldLogFile=/var/log/slurm/slurmctld.log
SlurmdDebug=3
SlurmdLogFile=/var/log/slurm/slurmd.log
JobCompType=jobcomp/filetxt
JobCompLoc=/var/log/slurm/jobcomp.log
#
# ACCOUNTING
JobAcctGatherType=jobacct_gather/linux
JobAcctGatherFrequency=30
#
AccountingStorageType=accounting_storage/slurmdbd
AccountingStorageHost=slurmdbd
AccountingStoragePort=6819
AccountingStorageLoc=slurm_acct_db
#AccountingStoragePass=
#AccountingStorageUser=
#
# COMPUTE NODES
NodeName=c1 Procs=2 Sockets=2 CoresPerSocket=1 RealMemory=6800 State=UNKNOWN
NodeName=c2 Procs=2 Sockets=2 CoresPerSocket=1 RealMemory=6800 State=UNKNOWN
#
# PARTITIONS
PartitionName=normal Default=yes Nodes=c1 Shared=YES State=UP
PartitionName=nextflow Nodes=c2 Shared=YES State=UP
PartitionName=chewBBACA Nodes=c1 Shared=YES State=UP QOS=chewbbaca
```

Once the server node has the slurm.conf correctly, we need to send this file to the other compute nodes.

```
# Example transfer to the slurm compute-1
scp slurm.conf root@compute-1:/etc/slurm/slurm.conf
```

Example *slurmdbd.conf*

```
#
# Example slurmdbd.conf file.
#
# See the slurmdbd.conf man page for more information.
#
# Archive info
#ArchiveJobs=yes
#ArchiveDir="/tmp"
#ArchiveSteps=yes
#ArchiveScript=
#JobPurge=12
#StepPurge=1
#
# Authentication info
AuthType=auth/munge
#AuthInfo=/var/run/munge/munge.socket.2
#
# slurmDBD info
```

(continues on next page)

(continued from previous page)

```
DbdAddr=slurmdbd
DbdHost=slurmdbd
#DbdPort=6819
SlurmUser=slurm
#MessageTimeout=300
DebugLevel=4
#DefaultQOS=normal,standby
LogFile=/var/log/slurm/slurmdbd.log
PidFile=/var/run/slurmdbd/slurmdbd.pid
#PluginDir=/usr/lib/slurm
#PrivateData=accounts,users,usage,jobs
#TrackWCKey=yes
#
# Database info
StorageType=accounting_storage/mysql
StorageHost=mysql
StorageUser=slurm
StoragePass=password
StorageLoc=slurm_acct_db
```

Now, we will configure the server Master node. We need to make sure that the server has all the right configurations and files.

```
# Check for log files existence and permissions
mkdir /var/spool/slurmctld
chown slurm: /var/spool/slurmctld
chmod 755 /var/spool/slurmctld
touch /var/log/slurmctld.log
chown slurm: /var/log/slurmctld.log
touch /var/log/slurm_jobacct.log /var/log/slurm_jobcomp.log
chown slurm: /var/log/slurm_jobacct.log /var/log/slurm_jobcomp.log
```

We also need to configure all the compute nodes. We need to make sure that all the compute nodes have the right configurations and files.

```
# Check for log files existence and permissions
mkdir /var/spool/slurmd
chown slurm: /var/spool/slurmd
chmod 755 /var/spool/slurmd
touch /var/log/slurmd.log
chown slurm: /var/log/slurmd.log
```

Use the following command to make sure that slurmd is configured properly on the compute machines.

```
sudo /etc/init.d/slurmd
```

Use the following command to launch the slurmdbd on the server.

```
sudo /etc/init.d/slurmdbd
```

Use the following command to launch the slurm controller on the master server.

```
sudo /etc/init.d/slurmcltd
```

1.6.3 Testing SLURM

To display the compute nodes use the following.

```
scontrol show nodes
```

1.7 Frontend Server

The Frontend server of the INNUENDO Platform is the application responsable for serving the static files to the user, interact with the potsgreSQL database, and send requests to the Controller server to submit jobs.

1.7.1 Installation

Good practice to install application specific dependencies is to first create a virtual environment, which will aggregate all the required dependencies for a specific application.

Because of that, the first thing to do is to install python virtualenv.

```
sudo apt-get install python-virtualenv
```

The code for the Frontend server is located at github and can be obtained using git.

```
git clone https://github.com/bfrgoncalves/INNUENDO_REST_API.git
```

To create the virtual environment, run the application inside the INNUENDO_REST_API folder.

```
cd INNUENDO_REST_API

# Create virtual environment
virtualenv flask
```

1.7.2 requirements.txt

The requirements.txt file is the file with all the required python dependencies for the application. To install them, run the following command inside the INNUENDO_REST_API folder.

```
flask/bin/pip install -r requirements.txt
```

Due to some lack of some dependencies, you might also need to install the following python packages described into the following links:

```
https://stackoverflow.com/questions/11618898/pg-config-executable-not-found
https://stackoverflow.com/questions/28253681/you-need-to-install-postgresql-server-
↪dev-x-y-for-building-a-server-side-extensi
https://stackoverflow.com/questions/23937933/could-not-run-curl-config-errno-2-no-
↪such-file-or-directory-when-installing
https://stackoverflow.com/questions/21530577/fatal-error-python-h-no-such-file-or-
↪directory
http://thefourtheye.in/2013/04/20/installing-python-ldap-in-ubuntu/
```

1.7.3 Bower Components

Bower is a package manager used to fetch all the client-side components required to create the user interface. It requires nodeJS for the installation so we need to install nodeJS before installing Bower and the client-side dependencies.

```
# Get nodeJS and install
curl -sL https://deb.nodesource.com/setup_6.x | sudo -E bash -
sudo apt-get install -y nodejs

# Install Bower
npm install -g bower
```

Install Bower components by running `bower install` inside the `INNUENDO_REST_API/app` folder.

1.7.4 Running the APP

To run the application, we first need to add the allegrograph client location to the path. To do it, install the Allegrograph client and run the following command.

```
export PYTHONPATH=/full/path/for/agraph-6.2.1-client-python/src/
```

Then, we need to run the *worker.py*‘ to allow classification and to send requests to PHYLOViZ Online and we need to run the `run.py` to launch the INNUENDO_REST_API application.

```
cd /path/to/INNUENDO_REST_API
./worker.py &
./run.py
```

1.8 Controller

Good practice to install application specific dependencies is to first create a virtual environment, which will aggregate all the required dependencies for a specific application.

Because of that, the first thing to do is to install python virtualenv.

```
sudo apt-get install python-virtualenv
```

The code for the Frontend server is located at github and can be obtained using git.

```
git clone https://github.com/bfrgoncalves/INNUENDO_PROCESS_CONTROLLER.git
```

To create the virtual environment, run the application inside the INNUENDO_PROCESS_CONTROLLER folder.

```
cd INNUENDO_PROCESS_CONTROLLER

# Create virtual environment
virtualenv flask
```

1.8.1 requirements.txt

The requirements.txt file is the file with all the required python dependencies for the application. To install them, run the following command inside the INNUENDO_PROCESS_CONTROLLER folder.

```
flask/bin/pip install -r requirements.txt
```

Due to some lack of some dependencies, you might also need to install the following python packages described into the following links:

```
https://stackoverflow.com/questions/12982486/glib-compile-error-ffi-h-but-libffi-is-
↪installed
https://stackoverflow.com/questions/22414109/g-error-trying-to-exec-cc1plus-execvp-no-
↪such-file-or-directory
```

1.8.2 Running the APP

To run the application, we first need to add the allegrograph client location to the path. To do it, install the Allegrograph client and run the following command.

```
export PYTHONPATH=/full/path/for/agraph-6.2.1-client-python/src/
```

Then, we need to run the `run.py` to launch the INNUENDO_PROCESS_CONTROLLER application.

```
./run.py
```

1.9 Nextflow

Nextflow is a workflow manager that enables scalable and reproducible scientific workflows using software containers. An overview of how to install and its requirements can be found on they documentation.

https://www.nextflow.io/docs/latest/index.html

However, for a simple installation, you can simply run the following commands.

```
wget -qO- https://get.nextflow.io | bash
```

This will install nextflow on the current directory and now you will need to add it to the path. Can simply move the nextflow executable to the /usr/local/bin

```
mv nextflow /usr/local/bin
```

You can now execute nextflow pipelines.

1.10 FlowCraft

Flowcraft is used in the INNUENDO Platform as the pipeline builder, which generates the pipelines according to the available protocols. Besides that, the flowcraft web-application is also used for pipeline process inspection and visualization of reports.

1.10.1 Installation

For the pipeline builder installation, check the Flowcraft [documentation](https://flowcraft.readthedocs.io/en/latest/?badge=latest)

For the install the Flowcraft webapp installation for pipleine inspection report visualization, follow the bellow steps:

```
# Clone Flowcraft webapp repository
git clone https://github.com/assemblerflow/flowcraft-webapp.git && cd flowcraft-webapp

# Install requirements (pipenv and >=python3.6 is required)
cd flowcraft-webapp
pipenv install --system --deploy --ignore-pipfile

# Install frontend dependencies
cd flowcraft-webapp && yarn install --network-timeout 1000000 && exit

# Construct required databases databases (postgreSQL is required)
python3 manage.py makemigrations
python3 manage.py migrate

# Build frontend required file
yarn run build

# Lauch the application
python3 manage.py runserver 0.0.0.0:6000
```

To configure the service, checkout how to do it by going here.

1.11 Docker-Compose

Docker-compose and the use of Docker allows running all the required INNUENDO Platform components in a controller environment (containers) in a very simple way.

Since it uses the docker-images as built using the developed Dockerfiles that act as a recipe for the installation of all components, it releases that burden from the user.

1.11.1 Installation

For the docker-compose version of the INNUENDO Platform you will need to install the following software.

- Docker
- Docker-Compose

On Ubuntu

First, add the GPG key for the official Docker repository to the system.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

Add the Docker repository to APT sources.

```
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
↪$(lsb_release -cs) stable"
```

Next, update the package database with the Docker packages from the newly added repo.

```
sudo apt-get update
```

Install Docker.

```
sudo apt-get install -y docker-ce
```

Docker should now be installed, the daemon started, and the process enabled to start on boot. Check that it's running.

```
sudo systemctl status docker
```

Next we will install docker-compose. We will check the current release and if necessary, update it in the command below.

```
sudo curl -L https://github.com/docker/compose/releases/download/1.18.0/docker-
↪compose-`uname -s`-`uname -m` -o /usr/local/bin/docker-compose
```

Next we will set the permissions.

```
sudo chmod +x /usr/local/bin/docker-compose
```

Then we can verify that the installation was successful by checking the version.

```
docker-compose --version
```

On Windows and Mac

Install the executables from the docker-compose page.

https://docs.docker.com/compose/install/

1.11.2 Configuration

Each component of the INNUENDO Platform can be configured by modifying its configuration file. Configuration files are located at `configs/` and are files required for the Platform to work.

NOTE: Modifying these files might lead to corruption of the application. Proceed with care.

Each file belonging to each component is described bellow.

Frontend Server

The Frontend server has one configuration file located at `configs/app/config_frontend.py` that has a set of variables required for this module to work in cooperation with the process controller.

Below defaults are for the docker-compose version.

`FRONTEND_IP` IP address of the machine, default: web

`phyloviz_root` Root address of PHYLOViZ Online. default: http://web:82

`AGRAPH_IP` AllegroGraph server IP adress. default: web

`CURRENT_ROOT` Current address of the frontend application. default: http://'+FRONTEND_IP+'/app

`JOBS_IP` INNUENDO Process Controller IP address. default: web

`JOBS_ROOT` Job submission route. default: http://'+JOBS_IP+'/jobs/'

`FILES_ROOT` Route to get information about fastq files. default:http://'+JOBS_IP+'/jobs/fastqs/'

`REPORTS_URL` Reports application route. default: "http://localhost/reports"

SECRET_KEY Secret key for flask-security hash.

SECURITY_PASSWORD_HASH Flask-security type of hash used.

SECURITY_PASSWORD_SALT Flaks-security salt used.

ADMIN_EMAIL Email of the platform administrator. default: innuendo@admin.com

ADMIN_NAME Administrator name. default: Admin

ADMIN_USERNAME ADministrator username. default: innuendo_admin

ADMIN_PASS Administrator password.

ADMIN_GID Group identifier for admins. default: 501

REDIS_URL Redis queue URL. default: redis://redis:6379

SECURITY_REGISTERABLE Allow Flask-security view to register. default: False

SECURITY_RECOVERABLE Allow Flask-security view to recover password. default: True

SECURITY_CHANGEABLE Allow Flask-security view to change password. default: True

SECURITY_FLASH_MESSAGES SHow Flask-security messages. default: True

FAST_MLST_PATH Path for fast-mlst application used for profile classification and search. default: /Frontend/fast-mlst

NEXTFLOW_TAGS Currently available FlowCraft tags. More information on FlowCraft documentation.

DATABASE_USER User owner of the postgreSQL database. default: innuendo

DATABASE_PASS Password of the postgreSQL user. default: innuendo_database

database_uri URI for the wgMLST profile database. default: 'postgresql://'+DATABASE_USER+':'+DATABASE_PASS + '@db_mlst/mlst_database'

innuendo_database_uri URI for the innuendo database. default: 'postgresql://'+DATABASE_USER+':'+DATABASE_PASS+'@db_innuendo/innuendo'

SQLALCHEMY_BINDS Databases that bind to SQLAlchemy.

SQLALCHEMY_MIGRATE_REPO Location to store and update database files. default: os.path.join(basedir, 'db_repository')

SQLALCHEMY_TRACK_MODIFICATIONS Track database modification. default: True

WTF_CSRF_ENABLED Enable CSRF. default: False

app_route Application entry route. default: '/app'

LDAP_PROVIDER_URL LDAP client IP definition. default: LDAP_IP

LDAP_PROTOCOL_VERSION LDAP protocol version. default: 3

baseDN Base repository reference. default: dc=innuendo,dc=com

LOGIN_METHOD Platform login method. Used to distinguish between LDAP authentication and single user authentication used in the docker version. default: None

LOGIN_GID Login group identifier. Used in case of docker version. default: 501

LOGIN_HOMEDIR Single user home directory. Used in case of docker version. default: /INNUENDO/

LOGIN_USERNAME Single user username. Used in case of docker version. default: innuendo_user

LOGIN_PASSWORD Single user password. Used in case of docker version. default: innuendo_user

LOGIN_EMAIL Single user email. Used in case of docker version. default: innuendo@innuendo.com

ALL_SPECIES All supported species. default: [“E.coli”,”Yersinia”,”Campylobacter”,”Salmonella”]

allele_classes_to_ignore chewBBACA report on profile to replace with 0.

wg_index_correspondece Path to the wg index file used by fast-mlst for profile search up to x differences. Example: {“E.coli”: “/INNUENDO/inputs/indexes/ecoli_wg”}

core_index_correspondece Path to the core index file used by fast-mlst for profile search up to x differences. Example: {“E.coli”: “/INNUENDO/inputs/indexes/ecoli_core”}

wg_headers_correspondece Path to the list of the wg loci for each species. Example: {“E.coli”: “/INNUENDO/inputs/core_lists/ecoli_headers_wg.txt”}

core_headers_correspondece Path to the list of the core loci for each species. Example: {“E.coli”: “/INNUENDO/inputs/core_lists/ecoli_headers_core.txt”}

core_increment_profile_file_correspondece Location of the file with the core profiles for each species. Used to contruct the search index. Example: {“E.coli”: “/INNUENDO/inputs/indexes/ecoli_core_profiles.tab”}

wg_increment_profile_file_correspondece Location of the file with wg profiles for each species. Used to contruct the search index. Example: {“E.coli”: “/INNUENDO/inputs/indexes/ecoli_wg_profiles.tab”}

classification_levels Classification levels for each specie. Number of profile differences. Example: {“E.coli”: [8, 112, 793]}

AG_REPOSITORY Name of the AllegroGraph repository. default: innuendo

AG_USER AllegroGraph user. default: innuendo

AG_PASSWORD AllegroGraph password. default: innuendo_allegro

Controller Server

The Controller server has one configuration file located at `configs/app/config_process.py` that has a set of variables required for this module to work in cooperation with the frontend and the workflow managers.

Below defaults are for the docker-compose version.

REDIS_URL Redis queue URL. default: redis://redis:6379

ASPERAKEY Aspera key location. default: ~/.aspera/connect/etc/asperaweb_id_dsa.openssh

FTP_FILES_FOLDER Location of the files folder in relation to the user home directory. default: ftp/files

NEXTFLOW_RESOURCES Specifications of each nextflow process. Can be used to specify each parameter of any given process. Example: { “integrity_coverage”:{“memory”: r“‘2GB’”,”cpus”: “1”}

SERVER_IP IP address of the machine. default: web

FRONTEND_SERVER_IP IP address of the frontend server. default: web

DEFAULT_SLURM_CPUS Default SLURM CPUs used when a process is not specified. default: 8

NEXTFLOW_PROFILE Nextflow profile to use. Those are specified in the FlowCraft software. default: desktop

NEXTFLOW_GENERATOR_PATH Location of the FlowCraft software executable. default: /Controller/flowcraft/flowcraft/flowcraft.py

NEXTFLOW_GENERATOR_RECIPE FlowCraft recipe to use. It defines the set of processes that can be used and their relationships. default: innuendo

FASTQPATH Location of the fastq files in the user directory structure. Used by FlowCraft to search for paired end reads. default: “data/*_{1,2}.*”

JOBS_ROOT_SET_OUTPUT Route used to set the output status of processes. Example: http://+SERVER_IP+/jobs/setoutput/

JOBS_ROOT_SET_REPORT Route used to set the reports and store them on the database. Example: http://+FRONTEND_SERVER_IP+/app/api/v1.0/jobs/report/

CHEWBBACA_PARTITION Partition name used by SLURM to launch chewBBACA processes. Can only run one chewBBACA at a time. default: chewBBACA

CHEWBBACA_SCHEMAS_PATH Location of the chewBBACA schemas. default: /INNUENDO/inputs/schemas

CHEWBBACA_TRAINING_FILE Location of prodigal training files for each specie. Example: { "E.coli": "/INNUENDO/inputs/prodigal_training_files/prodigal_training_files/Escherichia_coli.trn", }

SEQ_FILE_O SeqTyping FILE_O location. default: {"E.coli": "/INNUENDO/inputs/serotyping_files/escherichia_coli/1_O_type.fasta"}

SEQ_FILE_H Seqtyping FILE_H location. default: {"E.coli": "/INNUENDO/inputs/serotyping_files/escherichia_coli/2_H_type.fasta"}

wg_index_correspondece Path to the wg index file used by fast-mlst for profile search up to x differences. Example: {"E.coli": "/INNUENDO/inputs/indexes/ecoli_wg"}

core_index_correspondece Path to the core index file used by fast-mlst for profile search up to x differences. Example: {"E.coli": "/INNUENDO/inputs/indexes/ecoli_core"}

wg_headers_correspondece Path to the list of the wg loci for each species. Example: {"E.coli": "/INNUENDO/inputs/core_lists/ecoli_headers_wg.txt"}

core_headers_correspondece Path to the list of the core loci for each species. Example: {"E.coli": "/INNUENDO/inputs/core_lists/ecoli_headers_core.txt"}

core_increment_profile_file_correspondece Location of the file with the core profiles for each species. Used to contruct the search index. Example: {"E.coli": "/INNUENDO/inputs/indexes/ecoli_core_profiles.tab"}

wg_increment_profile_file_correspondece Location of the file with wg profiles for each species. Used to contruct the search index. Example: {"E.coli": "/INNUENDO/inputs/indexes/ecoli_wg_profiles.tab"}

AG_REPOSITORY AllegroGraph repository name. default: innuendo

AG_USER AllegroGraph username. default: innuendo

AG_PASSWORD AllegroGraph user password. default: innuendo_allegro

Flowcraft Configuration

The Flowcraft webapp application has two configuration files located at `configs/flowcraft` that has a set of variables required for this module to work in cooperation with the frontend.

Below are the defaults for the docker-compose version.

reportsRoute Route location to fetch for reports. default: http://localhost/reports

1.11.3 Running the INNUENDO Platform

Retrieving the docker-compose version

To launch the docker-compose version of the INNUENDO Platform, first need to get the INNUENDO_docker repository from github that has all the required Dockerfiles and structures for communication between the containers and the user file system.

```
git clone https://github.com/bfrgoncalves/INNUENDO_docker.git
```

Launching the application

Running the INNUENDO Platform is very simple. You can lauch it with a single command.

```
# Access the INNUENDO docker repository
cd </path/to/INNUENDO_docker>

# Launch the application
docker-compose up
```

The last command will pull all the required images first then it will launch all the Docker containers. They will will communicate between each other by a docker network that is built by default with docker-compose.

Downloading legacy data and building profile databases

The application provides a script to download all the required files to perform comparisons with some already publicly available strains. This is made through the download of the following data available here:

- chewBBACA schemas
- Legacy strain metadata (for each species)
- Legacy strain profiles (for each species)
- Serotyping files
- Prodigal training files

These data will be available under `./inputs` and will be mapped to the docker containers running the application.

The script also build the required files for a rapid comparison between profiles using fast-mlst and populates the `mlst_database`.

To run the script, type the following command:

```
# Enter repository directory
cd <innuendo_docker_directory>/build_files

# Run script to get legacy input files
./get_inputs.sh
```

These steps might take up to 1h depending on the available internet connection and the host machine.

Loading data from a pre-defined backup

We offer an option to load a predefined set of protocols and workflows, together with test projects and strains. Currently, this option is only available for machines with **above 8 cpus and 8gb of RAM**. This is due to the backup expecting at least those resources for at least one of the predefined protocols.

To load the predefined data, do the following:

```
# Enter the build_files directory
cd <innuendo_docker_directory>/build_files

# Run the script to load the data
./init_8cpu_components.sh
```

NOTE: The above script will delete ALL data available in the AllegroGraph database and INNUENDO general database. It will then replaced by the predefined data.

1.11.4 Mapping data into the Docker containers

To map data between the user filesystem and the containers, docker-compose already has a directive to deal with that action.

Inside the *docker-compose.yml* you got all the required attributes to launch the container and the interaction between other containers.

Below is described the directives used to launch a service in docker-compose.

```
# Service for the INNUENDO frontend. Requires the config files for the
# application and mapping of the fastq files
frontend:
    # this service uses the dockerfile inside the Frontend directory
    build: ./components/Frontend/
    # Allow run services inside as root
    privileged: true
    # Allow restart on failure
    restart: on-failure
    # Directive to map files and folders to the container. In this case,
    all files before : are files in the user file system. The files after
     : are the location of those files in the container.
    volumes:
      - ./configs/app/config_frontend.py:/Frontend/INNUENDO_REST_API/config.py
      - user_data:/INNUENDO
      - ./inputs/fastq:/INNUENDO/ftp/files
      - ./inputs/v1/classifications:/INNUENDO/inputs/v1/classifications
      - ./inputs/v1/core_lists:/INNUENDO/inputs/v1/core_lists
      - ./inputs/v1/indexes:/INNUENDO/inputs/v1/indexes
      - ./inputs/v1/legacy_metadata:/INNUENDO/inputs/v1/legacy_metadata
      - ./inputs/v1/legacy_profiles:/INNUENDO/inputs/v1/legacy_profiles
      - singularity_cache:/mnt/singularity_cache
    # Ports mapping between container and host
    ports:
      - "5000:5000"
    # Depends on other docker-compose services to work
    depends_on:
      - "allegro"
      - "db_innuendo"
      - "db_mlst"
      - "web"
    # Arguments to give to the docker-entrypoint.sh
    command: ["init_allegro", "build_db", "init_app"]
```

As seen above, the files can be mapped with the volumes directive.

Fastq files from the user must be placed into the `inputs/fastq` **folder to be linked with the INNUENDO Platform docker version.**

1.11.5 Backing up/ Build data

We provide a series of scripts to backup/build all the required databases used in the docker-compose version of the INNUENDO Platform. These files are located at inside the images and need to be triggered after the application is running. This is made using the `docker exec` command on an already running container.

Backing up/ Build postgreSQL databases

There are four postgreSQL databases used in the INNUENDO Platform that can be backed up: `innuendo`, `mlst_database`, `assemblerflow`, and `phyloviz`.

All databases backups can be made using a single command for each database.

```
# Execute script on frontend container to backup database
# Information on database, username and pass are located in the
# docker-compose.yml file
docker exec innuendo_docker_frontend_1 backup_dbs.sh backup <database> <username>
↪<pass> <backup_file_name>
```

The build command to restore a database to a given backup state is very similar to the above.

```
# Execute script on frontend container to build database
docker exec innuendo_docker_frontend_1 backup_dbs.sh build <database> <username>
↪<pass> <backup_file_name>
```

Backing up/ Build AllegroGraph databases

Other database type used in the INNUENDO Platform is a triplestore and it is also required for the application to retrive to a given state if required.

To backup AllegroGraph, it is only required to run a single command

```
# Execute script on frontend container to backup allegrograph
# Information on database, username and pass are located in the
# docker-compose.yml file
docker exec innuendo_docker_frontend_1 build_allegro.py backup <backup_file_name>
```

The build command is similar to the above and is required to move the application to a given state.

```
# Execute script on frontend container to backup allegrograph
docker exec innuendo_docker_frontend_1 build_allegro.py build <backup_file_name>
```

1.11.6 Customizing Entrypoints

Entrypoints are the files run on container creation with a series of predefined commands.

On each `component/` folder of the application you have an `entrypoint.sh` file and a Dockerfile.

By modifying the commands inside the `entrypoint.sh` you can change the default behaviour when the container for that component launches.

1.11.7 Useful docker commands

Bellow are some docker commands that might be useful to interact with the containers.

Show active containers.

```
docker-compose ps
```

Enter container.

```
docker exec -it container_name bash
```

List virtual volumes.

```
docker volume ls
```

List images.

```
docker images
```

Remove images

```
docker rmi image_name
```

1.12 Set a new Species

The INNUENDO Platform is **species dependent**. Which means that any project, protocol or workflow needs to be associated with a species. The scope of the INNUENDO Project was to develop analysis strategies from 4 target species: *Escherichia coli*, *Yersinia enterocolitica*, *Salmonella enterica* and *Campylobacter jejuni*. However, the platform is scalable to add any other species upon some configuration. In this example we are going to exemplify on how to add **speciesA**.

NOTE: Most of the modifications required are in the INNUENDO_REST_API application.

1.12.1 1 - Add a new database model

Each species in the INNUENDO Platform has a dedicated wgMLST profile database. As so, a new model for it needs to be added inside the *app/models/models.py* file of the *INNUENDO_REST_API* app.

```
# Example of adding species A. Inside models.py file near the other mlst
# database classes

class SpeciesA(db.Model):
    """
    Defines the species specific storage of profiles and its classification.
    Salmonella specification.
    """

    # Name of the database table
    __tablename__ = "speciesA"

    # The name of the mlst_database
    __bind_key__ = 'mlst_database'

```

(continues on next page)

(continued from previous page)

```
    # Required fields on each wgMLST species database
    id = db.Column(db.Integer(), primary_key=True)
    name = db.Column(db.String(255), unique=True)
    version = db.Column(db.String(255))
    # Platform classifiers
    classifier_l1 = db.Column(db.String(255))
    classifier_l2 = db.Column(db.String(255))
    classifier_l3 = db.Column(db.String(255))
    allelic_profile = db.Column(JSON)
    strain_metadata = db.Column(JSON)
    # Tell if it is legacy or from the platform
    platform_tag = db.Column(db.String(255))
    timestamp = db.Column(db.DateTime)
```

This new model needs to be loaded with *manage.py* in case of installation from source. In case of the docker-compose verison, it will be loaded automatically on start.

1.12.2 2 - Import model on app_configuration.py

The model needs then to be imported to be used by the application. This can be made by importing it at *app/app_configuration.py* of the *INNUENDO_REST_API* app. The *species_correspondece* dictionary needs also to be updated to allow association of the models with a key.

```
# Example of adding speciesA to the model imports at app/app_configuration.py
from app.models.models import Ecoli, Yersinia, Salmonella, Campylobacter, SpeciesA

# Change the species_correspondece object to associate model with a key
database_correspondece = {
    "E.coli": Ecoli,
    "Yersinia": Yersinia,
    "Salmonella": Salmonella,
    "SpeciesA": SpeciesA
}
```

1.12.3 3 - Update the config.py files

The *config.py* files need also to be updated in order for the application to know which species should use, the classification levels, and which files use for wgMLST database. These modifications are required on both *INNUENDO_REST_API* and *INNUENDO_PROCESS_CONTROLLER*.

Updating config.py on INNUENDO_REST_API application

```
# Example of config.py updates for speciesA

# Add speciesA to the list with all the available species
# NOTE: the name needs to be the same as the key used in the
# database_correspondece on step 2
ALL_SPECIES = [
    "E.coli",
    "Yersinia",
    "Campylobacter",
    "Salmonella",
    "SpeciesA"
```

(continues on next page)

(continued from previous page)

```
]

# Add the  Association between species ID in the platform with the species name
SPECIES_CORRESPONDENCE = {
    "E.coli": "Escherichia coli",
    "Yersinia": "Yersinia enterocolitica",
    "Salmonella": "Salmonella enterica",
    "Campylobacter": "Campylobacter jejuni"
    "SpeciesA": "Species A real name"
};

# Add the wgMLST fast-mlst index file correspondence
wg_index_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_wg",
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_wg",
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_wg",
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_wg"
    }
}

# Add Path to the core index file used by fast-mlst for profile search up to x
# differences
core_index_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_core",
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_core",
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_core",
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_core"
    }
}

# Add Path to the list of the wg loci for each species
wg_headers_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/core_lists/ecoli_headers_wg.txt",
        "Yersinia": "/INNUENDO/inputs/v1/core_lists/yersinia_headers_wg.txt",
        "Salmonella": "/INNUENDO/inputs/v1/core_lists/salmonella_headers_wg.txt",
        "SpeciesA": "/INNUENDO/inputs/v1/core_lists/speciesA_headers_wg.txt"
    }
}

# Add Path to the list of the core loci for each species
core_headers_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/core_lists/ecoli_headers_core.txt",
        "Yersinia": "/INNUENDO/inputs/v1/core_lists/yersinia_headers_core.txt",
        "Salmonella": "/INNUENDO/inputs/v1/core_lists/salmonella_headers_core.txt",
        "SpeciesA": "/INNUENDO/inputs/v1/core_lists/speciesA_headers_core.txt"
    }
}

# Add Location of the file with the core profiles for each species. Used to
# contruct the search index
core_increment_profile_file_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_core_profiles.tab",
```

(continues on next page)

(continued from previous page)

```
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_core_profiles.tab",
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_core_profiles.tab",
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_core_profiles.tab"
    }
}

# Add Location of the file with wg profiles for each species. Used to contruct the
# search index
wg_increment_profile_file_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_wg_profiles.tab",
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_wg_profiles.tab",
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_wg_profiles.tab",
        "Campylobacter": "/INNUENDO/inputs/v1/indexes/campy_wg_profiles.tab"
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_wg_profiles.tab"
    }
}
```

Updating config.py on INNUENDO_PROCESS_CONTROLLER application

```
# Add chewBBACA prodigal training file if not assigned in the protocol
CHEWBBACA_TRAINING_FILE = {
    "E.coli": "/INNUENDO/inputs/prodigal_training_files/prodigal_training_files/
↪Escherichia_coli.trn",
    "Yersinia": "/INNUENDO/inputs/prodigal_training_files/prodigal_training_files/
↪Yersinia_enterocolitica.trn",
    "Campylobacter": "/INNUENDO/inputs/prodigal_training_files/prodigal_training_
↪files/Campylobacter_jejuni.trn",
    "Salmonella": "/INNUENDO/inputs/prodigal_training_files/prodigal_training_files/
↪Salmonella_enterica.trn"
    "SpeciesA": "/prodigal/training/file/location"
}

# Add name user for chewBBACA in case not assigned in the protocol
CHEWBBACA_CORRESPONDENCE = {
    "E.coli": "Escherichia coli",
    "Yersinia": "Yersinia enterocolitica",
    "Campylobacter": "Campylobacter jejuni",
    "Salmonella": "Salmonella enterica",
    "SpeciesA": "Species a"
}

# Add Torsten's mlst correspondence
MLST_CORRESPONDENCE = {
    "E.coli": "ecoli",
    "Yersinia": "yersinia",
    "Campylobacter": "campylobacter",
    "Salmonella": "senterica",
    "SpeciesA": "speciesa"

}

# Add the wgMLST fast-mlst index file correspondence
wg_index_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_wg",
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_wg",
```

(continues on next page)

(continued from previous page)

```
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_wg",
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_wg"
    }
}

# Add Path to the core index file used by fast-mlst for profile search up to x
# differences
core_index_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_core",
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_core",
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_core",
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_core"
    }
}

# Add Path to the list of the wg loci for each species
wg_headers_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/core_lists/ecoli_headers_wg.txt",
        "Yersinia": "/INNUENDO/inputs/v1/core_lists/yersinia_headers_wg.txt",
        "Salmonella": "/INNUENDO/inputs/v1/core_lists/salmonella_headers_wg.txt",
        "SpeciesA": "/INNUENDO/inputs/v1/core_lists/speciesA_headers_wg.txt"
    }
}

# Add Path to the list of the core loci for each species
core_headers_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/core_lists/ecoli_headers_core.txt",
        "Yersinia": "/INNUENDO/inputs/v1/core_lists/yersinia_headers_core.txt",
        "Salmonella": "/INNUENDO/inputs/v1/core_lists/salmonella_headers_core.txt",
        "SpeciesA": "/INNUENDO/inputs/v1/core_lists/speciesA_headers_core.txt"
    }
}

# Add Location of the file with the core profiles for each species. Used to
# contruct the search index
core_increment_profile_file_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_core_profiles.tab",
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_core_profiles.tab",
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_core_profiles.tab",
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_core_profiles.tab"
    }
}

# Add Location of the file with wg profiles for each species. Used to contruct the
# search index
wg_increment_profile_file_correspondece = {
    "v1": {
        "E.coli": "/INNUENDO/inputs/v1/indexes/ecoli_wg_profiles.tab",
        "Yersinia": "/INNUENDO/inputs/v1/indexes/yersinia_wg_profiles.tab",
        "Salmonella": "/INNUENDO/inputs/v1/indexes/salmonella_wg_profiles.tab",
        "Campylobacter": "/INNUENDO/inputs/v1/indexes/campy_wg_profiles.tab"
        "SpeciesA": "/INNUENDO/inputs/v1/indexes/speciesA_wg_profiles.tab"
    }
```

(continues on next page)

(continued from previous page)

```
}

# Update the expected genome size of SpeciesA
species_expected_genome_size = {
    "E.coli": "5",
    "Yersinia": "4.7",
    "Salmonella": "4.6",
    "Campylobacter": "1.6",
    "SpeciesA": "GenomeSize"
}
```

To know on how to create the required legacy database files, check the *Set legacy database* section.

1.13 Set a legacy profiles database

The INNUENDO Platform allows adding profiles already analysed to the wgMLST database for comparison. These profiles must have an associated metadata and the three level profile classification.

This can be made by updating the following files of the *INNUENDO_REST_API* application, at the *INNUENDO_REST_API/build_files* folder. The files are:

- **build_indexes.sh** - Gets profiles, metadata, and classification. It also adds the information to the new wgMLST database.
- **get_profiles_and_training.sh** - Gets the used wgMLST schema.

The above files should be changed to add according to the modifications required to insert the data inside the database. Check the documentation inside the above files for more information regarding each step.

An example of each of the input files can be found:

- Allelic profiles .tab file (Yersinia enterocolitica)
- Metadata file (Yersinia enterocolitica)
- List of schema core genes (Yersinia enterocolitica)
- Three level classification (Yersinia enterocolitica)

1.14 Set Protocols

In the INNUENDO Platform, protocols are the basic unit for running processes. They are the building blocks to construct Workflows, which can then be applied to strains in our projects.

Protocol creation is responsibility of the INNUENDO platform administrators.

Protocols are composed of a **Type**, the name of the **used Software**, a **Nextflow Tag**, **Parameters**, and a **Name**. Each protocol name *MUST* be unique.

1.14.1 Protocol Types

Protocol types are defined by NGSOnto and are a way of classifying the available protocols. Each type can have different attributes.

- de-novo assembly protocol

- Sequencing quality control protocol
- Allele Call Protocol
- sequencing Protocol
- DNA Extraction protocol
- Pathotyping Protocol
- Sequence cutting protocol
- mapping assembly protocol
- Filtering protocol
- Library Preparation Protocol

1.14.2 used Software

When creating a protocol, other field that needs to exist is the **used Software**. It is required for the Platform to know which software you are going to use on that protocol in case some extra steps are required after or before running it. The available tags are:

- reads_download
- seq_typing
- patho_typing
- integrity_coverage
- fastqc (fastqc_trimmomatic)
- true_coverage
- fastqc_2 (fastqc)
- integrity_coverage_2 (check_coverage)
- spades
- process_mapping
- pilon
- mlst
- sistr
- chewBBACA
- abricate

Each of these tags are closely related to the Nextflow Tags chosen. So, to have a good agreement between Software and Nextflow Tags, pair them together.

1.14.3 Nextflow Tags

Nextflow Tags are the specific names that FlowCraft requires to build Nextflow pipelines based on the available software at the INNUENDO Platform.

Below you have all the available Nextflow Tags retrieved from FLowCraft that can be used in the INNUENDO Platform:

```
=> reads_download
   input_type: accessions
   output_type: fastq
   dependencies: None
=> seq_typing
   input_type: fastq
   output_type: None
   dependencies: None
=> patho_typing
   input_type: fastq
   output_type: None
   dependencies: None
=> integrity_coverage
   input_type: fastq
   output_type: fastq
   dependencies: None
=> fastqc_trimmomatic
   input_type: fastq
   output_type: fastq
   dependencies: integrity_coverage
=> true_coverage
   input_type: fastq
   output_type: fastq
   dependencies: None
=> fastqc
   input_type: fastq
   output_type: fastq
   dependencies: None
=> check_coverage
   input_type: fastq
   output_type: fastq
   dependencies: None
=> spades
   input_type: fastq
   output_type: fasta
   dependencies: integrity_coverage
=> process_spades
   input_type: fasta
   output_type: fasta
   dependencies: None
=> assembly_mapping
   input_type: fasta
   output_type: fasta
   dependencies: None
=> pilon
   input_type: fasta
   output_type: fasta
   dependencies: assembly_mapping
=> mlst
   input_type: fasta
   output_type: fasta
   dependencies: None
=> abricate
   input_type: fasta
   output_type: None
   dependencies: None
=> chewbbaca
```

(continues on next page)

(continued from previous page)

```
   input_type: fasta
   output_type: None
   dependencies: None
=> sistr
   input_type: fasta
   output_type: None
   dependencies: None
```

1.14.4 Protocol Name

The protocol name is the identifier that will appear when choosing protocols to apply to a Workflow. Each protocol name **MUST** be unique. Also, try to make a reference for the nextflow tag used in the protocol name in order to establish a better organization regarding available protocols.

For more information regarding **FlowCraft**, checkout this link: https://assemblerflow.readthedocs.io/en/dev/index.html

1.15 Set Workflows

In the INNUENDO Platform, Workflows are the merge of one or more protocols to build a cascade of events to be applied to you strains. Their goal is to organize a group of software to be applied and you can then apply multiple workflows to a strain and build a pipeline according to their specific **Workflow dependencies**.

As for Protocols, Workflows also have a predefined set of attributes that need to be filled in order to be successfully applied to a strain. A workflow must have a **Name**, a **Dependency**, a **Type**, and the **Species** where that workflow will be available.

Workflows are **Species dependent** so you need to define the workflows that you want to make available for each species.

Workflow creation is responsibility of the INNUENDO platform administrators.

1.15.1 Workflow Name

Each workflow **MUST** have a name and it cannot be the same even across Species. The use of special characters are discouraged.

1.15.2 Workflow Dependency

Workflows can have input dependencies that are required to run them. Dependencies can be **Fastq** files, **Accession** numbers or any one of the already available workflows. These dependencies will them be used to guide the user when applying workflows to their strains.

1.15.3 Type

Workflows in the INNUENDO Platform are separated into two types: **Classifier** and **Procedure**.

- **Classifier** - Procedure to classify non-computing required processes. Used for classification of processes prior to data analysis. Not currently implemented in the INNUENDO Platform.

- **Procedure** - A procedure is a workflow that can be applied to a strain and run on the data associated to that strain.

Currently, only Procedures can be applied to strains.

1.15.4 Workflow Recipes

For the INNUENDO Platform, there are a set of Workflow recipes that can be constructed to run software on the strain data in the correct order. They depend on the created protocols which in the examples below they have the same name as their Nextflow Tags.

- **Reads Download:**
  - **Protocols (1):**
    - reads_download
- **Serotyping:**
  - **Protocols (1):**
    - seq_typing
- **Pathotyping:**
  - **Protocols (1):**
    - patho_typing
- **INNUca:**
  - **Protocols (10):**
    - integrity_coverage
    - fastqc_trimmomatic
    - true_coverage
    - fastqc
    - check_coverage
    - spades
    - process_spades
    - assembly_mapping
    - pilon
    - mlst
- **chewBBACA:**
  - **Protocols (1):**
    - chewbbaca
    - **Protocol Parameters:**
      - schema: chewbbaca_schema_folder_name
- **Abricate:**
  - **Protocols (1):**
    - abricate

- **SISTR:**
  - **Protocols (1):**
    - sistr

1.16 Backing up Data

Backing up data is an essential feature on every system and in the INNUENDO Platform that is no exception. As so, bellow we provide the required commands to backup all data on the system.

The INNUENDO Platform is composed of 3 databases: The **frontend database**, the **wgMLST database** and the **allegrograph database**. The first two are postgreSQL relational databases and the third a triplestore (graph based database).

1.16.1 Backing up postgreSQL databases

postgreSQL provides a built in tool for backing up its databases. It builds a file with all the instructions required to rebuild the database in other instance if required.

To backup the frontend database, run the following command on the machine running the service:

```
# This command will produce a new file called output_file.db that will
# have all the instructions to build the database. Replace
# <database_user> and <database_name> by the database owner and the wgMLST
# database name.
pg_dump -U <database_user> <database_name> > output_file_frontend.db
```

To backup the wgMLST database, run the following command on the machine running the service:

```
# This command will produce a new file called output_file.db that will
# have all the instructions to build the database. Replace
# <database_user> and <database_name> by the database owner and the wgMLST
# database name.
pg_dump -U <database_user> <database_name> > output_file_wgmlst.db
```

To restore the database, run the following command on the machine running the postgreSQL service.

```
# The text files created by pg_dump are intended to be read in by the psql program.
# Replace <database_user> and <database_name> by the database owner and database name.
psql -U <database_user> <database_name> < output_file.db
```

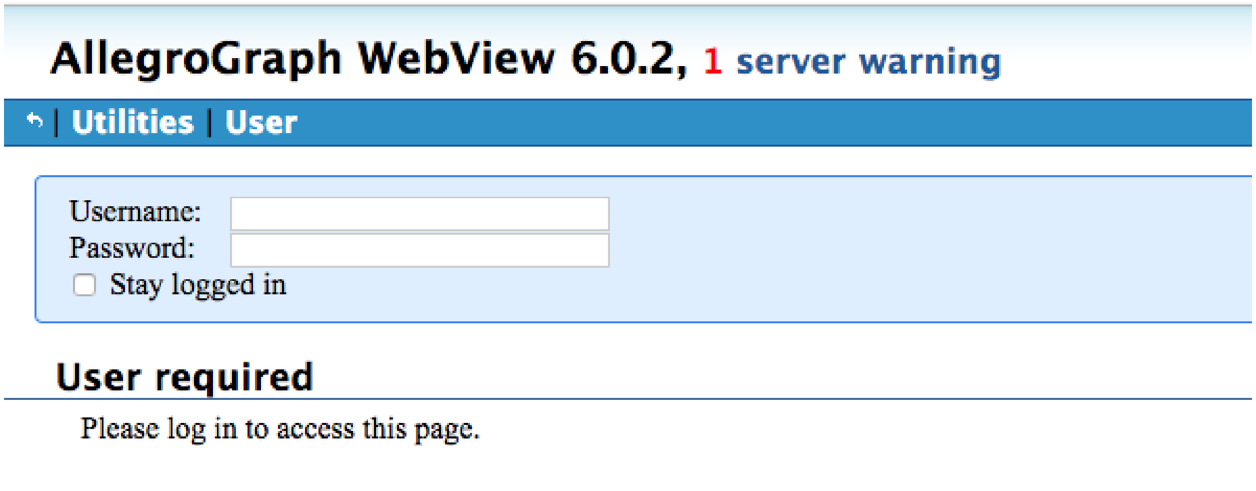
The INNUENDO Platform also provides a script for automatic backup of postgreSQL databases located inside the **build_files** directory inside **INNUENDO_REST_API**.

```
# Parameters
# mode: [backup, build]
# database: database_name
# postgresUser: Postgres username and owner of database
# postgresPass: Postgres password
# fileLocation: Location of output or input file (depening on the mode
backup_dbs.sh <mode> <database> <postgresUser> <postgresPass> <fileLocation>
```

1.16.2 Backing up AllegroGraph database

The AllegroGraph database is a different type of database. Is not a relational database. Instead, it stores relationships between objects in the form of a graph. It is used on the INNUENDO Platform as the backbone to get track of relationships between, projects, strains, workflows, processes and their outputs.

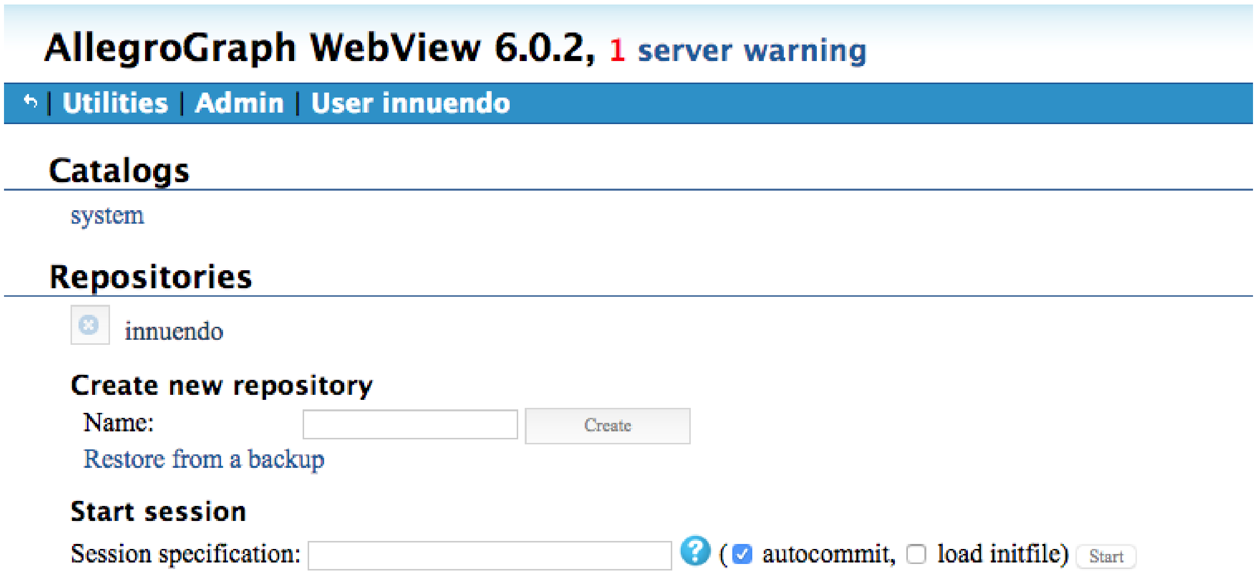
To backup the AllegroGraph database, we can use their web application. To do that, go to the defined configuration url for the AllegroGraph web application. There you will need to login as seen bellow with your AllegroGraph username and password.



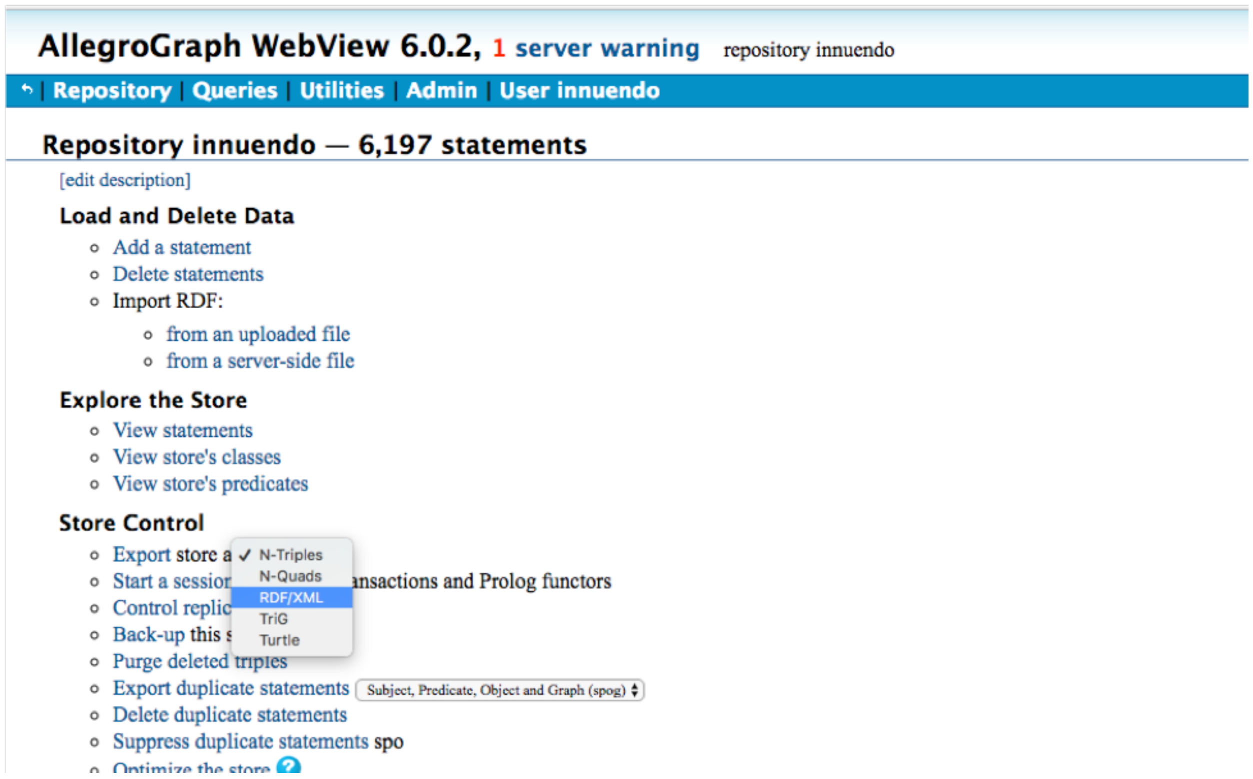
User required

Please log in to access this page.

After logging in, you will enter in a new page with information regarding the available repositories. You should see the already created repository for the INNUENDO Platform. In this case, it has the name *innuendo*.



After clicking on the desired repository, you can export the database by going to **Export store as** and select **RDF/XML**. This will create a file with the structure of the database that you can then load into AllegroGraph also using the web application.



AllegroGraph WebView 6.0.2, 1 server warning repository innuendo

↰ | Repository | Queries | Utilities | Admin | User innuendo

Repository innuendo — 6,197 statements

[edit description]

Load and Delete Data

- Add a statement
- Delete statements
- Import RDF:
  - from an uploaded file
  - from a server-side file

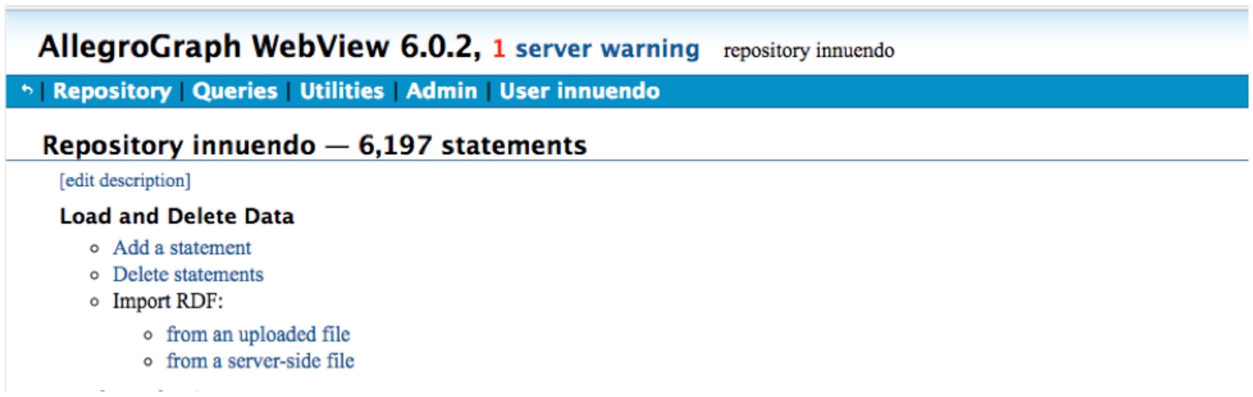
Explore the Store

- View statements
- View store's classes
- View store's predicates

Store Control

- Export store a
- Start a session ansactions and Prolog functors
- Control replic
- Back-up this s
- Purge deleted triples
- Export duplicate statements Subject, Predicate, Object and Graph (spog)
- Delete duplicate statements
- Suppress duplicate statements spo
- Optimize the store

To do that, on the same page as the Export, there is an option to **Import RDF**. Choose the option **from an uploaded file** and add one file obtained from the Export option. At the end you should get the repository restored.



AllegroGraph WebView 6.0.2, 1 server warning repository innuendo

↰ | Repository | Queries | Utilities | Admin | User innuendo

Repository innuendo — 6,197 statements

[edit description]

Load and Delete Data

- Add a statement
- Delete statements
- Import RDF:
  - from an uploaded file
  - from a server-side file

In addition to the previous steps, the INNUENDO Platform provides a programmatic way to backup and restore the AllegroGraph database using the script **build_allegro.py** located at the **build_files** directory inside **INNUENDO_REST_API**.

```
# Parameters
# mode: [backup, build]
# fileLocation: Location of the output or input file
#
# Steps
# Copy build_allegro.py to INNUENDO_REST_API since it requires to be run
on that location.
cp <INNUENDO_REST_API_location>/build_files/build_allegro.py <INNUENDO_REST_API_
↪location>/
# Add AllegroGraph client to the PYTHONPATH
```

(continues on next page)

(continued from previous page)

```
export PYTHONPATH="<INNUENDO_REST_API_LOCATION>/agraph-6.2.1-client-python/src"
# Run the script
flask/bin/python build_allegro.py <mode> <fileLocation>
```

1.16.3 Backing up Nextflow runs Data

All processes submitted to the INNUENDO Platform are managed by Nextflow and SLURM managers. Software runs that they manage are stored in the file system in directories structures and not in databases. As so, results derived directly from the software being run stay in those directory structures eg, raw reads, fasta files and other software outputs. Only post-processed selected data is sent to the INNUENDO Reports to be visualized.

Data from runs is stored by default in the */<usersStorage>/<user>/<jobs>*.

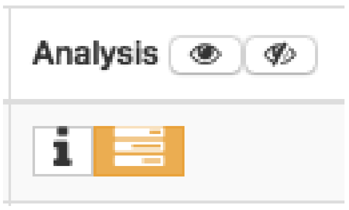
Inside each job folder you will have results and recipes to run the processes for each strain. Since each pipeline is associated with a strain in a given project, inside the *jobs* directory you will find folders with the structure *<project_id>-<pipeline_id>*. Inside those folder you can find other folder called *results* where all the relevant information regarding that pipeline is stored.

```
# Runs directory structure

- <usersStorage>
    - <user>
        - <jobs>
            - <project_id>-<pipeline_id>
                - <results>
                - <work>
                    - processes generated files
                - executor_command.sh -> To rerun pipeline
```

1.17 Inspecting Platform Logs

Admins of the INNUENDO Platform have some extra features to visualize logs for each process on every Project. When an admin enters a project he can visualize the logs by clicking on the *Information* button available for each strain on a project, in the *Analysis* column.



By clicking on that button the admin gets access to the information described bellow.

1.17.1 FlowCraft Build Log

The INNUENDO Platform builds the pipelines based on the FlowCraft software. It builds the nextflow files required by using the Nextflow Tags defined when creating the protocols as inputs.

Information regarding cpu usage, memory, and other directives can also be passed to FlowCraft when building the pipelines.

```
python3 /home/ubuntu/innuendo/flowcraft/flowcraft/flowcraft.py build -t reads_
↪download={'pid':1,'cpus':'2','memory':'\'4GB\''} integrity_coverage={'pid':2,'cpus':
↪'1','memory':'\'4GB\''} fastqc_trimmomatic={'pid':3,'cpus':'2','memory':'\'4GB\''}␣
↪true_coverage={'pid':4,'cpus':'2','memory':'\'4GB\''} fastqc={'pid':5,'cpus':'2',
↪'memory':'\'4GB\''} check_coverage={'pid':6,'cpus':'1','memory':'\'4GB\''} spades={
↪'pid':7,'scratch':'true','cpus':'4','memory':'\'4GB\''} process_spades={'pid':8,
↪'cpus':'1','memory':'\'4GB\''} assembly_mapping={'pid':9,'cpus':'2','memory':'\'4GB\
↪''} pilon={'pid':10,'cpus':'2','memory':'\'4GB\''} mlst={'pid':11,'version':
↪'tuberfree','cpus':'1','memory':'\'4GB\''} abricate={'pid':12,'cpus':'2','memory':'\
↪'4GB\''} chewbbaca={'pid':13,'queue':'\'chewBBACA\'','cpus':'8','memory':'\'4GB\''}␣
↪-o /mnt/innuendo_storage/users/bgoncalves/jobs/8-9/8_9.nf -r innuendo
[1;32m========= F L O W C R A F T =========
Build mode
version: 1.1.0
build: 20042018
=====================================[0m
[1;38mResulting pipeline string:
[0m
[1;38m reads_download={'pid':1,'cpus':'2','memory':'\'4GB\''} integrity_coverage={'pid
↪':2,'cpus':'1','memory':'\'4GB\''} fastqc_trimmomatic={'pid':3,'cpus':'2','memory':
↪'\'4GB\''} true_coverage={'pid':4,'cpus':'2','memory':'\'4GB\''} fastqc={'pid':5,
↪'cpus':'2','memory':'\'4GB\''} check_coverage={'pid':6,'cpus':'1','memory':'\'4GB\'
↪'} spades={'pid':7,'scratch':'true','cpus':'4','memory':'\'4GB\''} process_spades={
↪'pid':8,'cpus':'1','memory':'\'4GB\''} assembly_mapping={'pid':9,'cpus':'2','memory
↪':'\'4GB\''} pilon={'pid':10,'cpus':'2','memory':'\'4GB\''} mlst={'pid':11,'version
↪':'tuberfree','cpus':'1','memory':'\'4GB\''} ( abricate={'pid':12,'cpus':'2','memory
↪':'\'4GB\''} | chewbbaca={'pid':13,'queue':'\'chewBBACA\'','cpus':'8','memory':'\
↪'4GB\''} )
[0m
[1;38mChecking pipeline for errors...[0m
[1;38mBuilding your awesome pipeline...[0m
[1;38m     Successfully connected 13 process(es) with 1 fork(s) across 3 lane(s) ✓[0m
[1;38m     Channels set for init ✓[0m
[1;38m     Channels set for reads_download ✓[0m
[1;38m     Channels set for integrity_coverage ✓[0m
[1;38m     Channels set for fastqc_trimmomatic ✓[0m
[1;38m     Channels set for true_coverage ✓[0m
[1;38m     Channels set for fastqc ✓[0m
[1;38m     Channels set for check_coverage ✓[0m
[1;38m     Channels set for spades ✓[0m
[1;38m     Channels set for process_spades ✓[0m
[1;38m     Channels set for assembly_mapping ✓[0m
[1;38m     Channels set for pilon ✓[0m
[1;38m     Channels set for mlst ✓[0m
[1;38m     Channels set for abricate ✓[0m
[1;38m     Channels set for chewbbaca ✓[0m
[1;38m     Successfully set 10 secondary input(s) ✓[0m
[1;38m     Successfully set 3 secondary channel(s) ✓[0m
[1;38m     Finished configurations ✓[0m
[1;38m     Pipeline written into /mnt/innuendo_storage/users/bgoncalves/jobs/8-9/8_9.
↪nf ✓[0m
[1;32mDONE![0m
```

1.17.2 Platform Config

When running a pipeline using FlowCraft, there are some input variables required depending on the software used. Below is described the inputs required to run the pipeline built above.

```
params {
    accessions="/mnt/innuendo_storage/users/bgoncalves/jobs/8-9/accessions.txt"
    platformSpecies="Campylobacter"
    referenceFileO=""
    currentUserName="bgoncalves"
    schemaSelectedLoci="/home/ubuntu/innuendo/schemas/ccoli_cjejuni_Py3/listGenes.txt"
    currentUserId="4"
    projectId="8"
    asperaKey="/mnt/singularity_cache/shared_files/asperaweb_id_dsa.openssh"
    pipelineId="9"
    reportHTTP="http://192.168.1.10/app/api/v1.0/jobs/report/"
    chewbbacaTraining="/home/ubuntu/innuendo/prodigal_training_files/prodigal_
↪training_files/Campylobacter_jejuni.trn"
    schemaPath="/home/ubuntu/innuendo/schemas/ccoli_cjejuni_Py3"
    referenceFileH=""
    genomeSize="1.6"
    platformHTTP="http://192.168.1.11/jobs/setoutput/"
    sampleName="ERR2601756"
    species="Campylobacter jejuni"
    mlstSpecies="campylobacter"
    schemaCore="/home/ubuntu/innuendo/core_lists/core_lists/campy_headers_core.txt"
    chewbbacaJson=true
}
```

1.17.3 Nextflow Run Logs

After starting a run with Nextflow, it starting given the log above what processes are being submitted. Below is described the *nextflow log* that is provided by nextflow on every pipeline run.

```
N E X T F L O W  ~  version 0.30.1
Launching `/mnt/innuendo_storage/users/bgoncalves/jobs/8-9/8_9.nf` [agitated_lamarr] -
↪ revision: d8c53f4c58
WARN: It seems you never run this project before -- Option `-resume` is ignored
WARN: The config file defines settings for an unknown process: chewbbaca -- Did you␣
↪mean: chewbbaca_13?

============================================================
                F L O W C R A F T
============================================================
Built using flowcraft v1.1.0

 Input accessions             : 1
 Reports are found in         : ./reports
 Results are found in         : ./results
 Profile                      : incd

Starting pipeline at Thu Jun 14 09:56:45 UTC 2018

[warm up] executor > slurm
[5b/597bde] Submitted process > reads_download_1 (ERR2601756)
[64/4ace33] Submitted process > report (null)
```

(continues on next page)

(continued from previous page)

```
[13/550a1e] Submitted process > status (ERR2601756)
[1e/0cba15] Submitted process > integrity_coverage_2 (ERR2601756)
[70/e7768f] Submitted process > status (ERR2601756)
[ee/6719aa] Submitted process > report_coverage_2
[c1/689d2d] Submitted process > report (null)
[7f/34afa4] Submitted process > fastqc_3 (ERR2601756)
[e3/af508e] Submitted process > status (ERR2601756)
[eb/5e1ca3] Submitted process > report (null)
[bf/979467] Submitted process > fastqc_report_3 (ERR2601756)
[35/b57a70] Submitted process > report (null)
[de/2443d8] Submitted process > status (ERR2601756)
[11/80a9bc] Submitted process > trim_report_3
[b5/e9c230] Submitted process > compile_fastqc_status_3
[65/4b8bfe] Submitted process > trimmomatic_3 (ERR2601756)
[2e/9f4768] Submitted process > status (ERR2601756)
[cf/bae302] Submitted process > report (null)
[64/d1163d] Submitted process > true_coverage_4 (ERR2601756)
[73/7b60a2] Submitted process > status (ERR2601756)
[8c/572e1a] Submitted process > report (null)
[5a/b3b8d7] Submitted process > fastqc2_5 (ERR2601756)
[d6/1e6203] Submitted process > status (ERR2601756)
[1c/a1e215] Submitted process > report (null)
[44/9376fe] Submitted process > fastqc2_report_5 (ERR2601756)
[ea/a88007] Submitted process > report (null)
[48/26e31a] Submitted process > status (ERR2601756)
[78/747a8e] Submitted process > compile_fastqc_status2_5
[fd/e66412] Submitted process > integrity_coverage2_6 (ERR2601756)
[cf/110b82] Submitted process > report (null)
[90/2cbe38] Submitted process > report_coverage_2_6
[07/1ec1ad] Submitted process > status (ERR2601756)
[4f/35ef82] Submitted process > spades_7 (ERR2601756)
[7e/b41ef9] Submitted process > status (ERR2601756)
[dc/3e9b1c] Submitted process > report (null)
[ec/0b1648] Submitted process > process_spades_8 (ERR2601756)
[9a/474fed] Submitted process > status (ERR2601756)
[20/de7925] Submitted process > report (null)
[b4/797d97] Submitted process > assembly_mapping_9 (ERR2601756)
[e9/076467] Submitted process > status (ERR2601756)
[f7/b2a31c] Submitted process > report (null)
[41/0c0c46] Submitted process > process_assembly_mapping_9 (ERR2601756)
[35/79afe6] Submitted process > report (null)
[ef/d7dc5e] Submitted process > status (ERR2601756)
[98/d4720a] Submitted process > pilon_10 (ERR2601756)
[fb/c97d88] Submitted process > report (null)
[ff/ae8145] Submitted process > status (ERR2601756)
[e8/b64b34] Submitted process > mlst_11 (ERR2601756)
[f6/e6cc66] Submitted process > pilon_report_10 (ERR2601756)
[d0/2a9846] Submitted process > compile_mlst_11
[69/2c6096] Submitted process > report (null)
[c9/e8a0ed] Submitted process > status (ERR2601756)
[85/0f52a0] Submitted process > abricate_12 (ERR2601756 vfdb)
[d3/daa42d] Submitted process > abricate_12 (ERR2601756 virulencefinder)
[12/87469f] Submitted process > abricate_12 (ERR2601756 plasmidfinder)
[e5/86de26] Submitted process > abricate_12 (ERR2601756 card)
[83/083261] Submitted process > abricate_12 (ERR2601756 resfinder)
[5f/0d7e95] Submitted process > chewbbaca_13 (ERR2601756)
[84/0c1512] Submitted process > report (null)
```

(continues on next page)

(continued from previous page)

```
[26/eefb06] Submitted process > compile_pilon_report_10
[d6/488e5f] Submitted process > status (ERR2601756)
[a5/0b88b0] Submitted process > status (ERR2601756)
[ff/3ce639] Submitted process > report (null)
[3a/653f1d] Submitted process > status (ERR2601756)
[67/969e84] Submitted process > report (null)
[24/431fa8] Submitted process > report (null)
[76/4b245f] Submitted process > status (ERR2601756)
[7d/498efd] Submitted process > status (ERR2601756)
[22/351c99] Submitted process > report (null)
[59/b86166] Submitted process > report (null)
[1c/727625] Submitted process > status (ERR2601756)
[8d/6cc53d] Submitted process > process_abricate_12
[4f/e8a62d] Submitted process > report (null)
[72/9709c5] Submitted process > status (ERR2601756)
[fd/395422] Submitted process > compile_reports
[a0/cd14c7] Submitted process > compile_status_buffer (1)
[4f/3abd7f] Submitted process > compile_status
Completed at: Thu Jun 14 10:32:18 UTC 2018
Duration    : 35m 32s
Success     : true
Exit status : 0
```

Between *[]* is described on which folder inside the user jobs directory structure the data is being stored for that particular process. As so, the results from *reads_download_1* are being stored at */<usersStorage>/<user>/<jobs>/<project_id>-<pipeline_id>/work/5b/597bde*.

To visualize the specific log for that process we should go to the folder described above and check for the files *.command.log* and *.command.err*, which are the nextflow files generated with the outputs of a process.

1.18 Troubleshooting

In this section we are going to add some of the possible scenarios that can cause the admins to interact with the INNUENDO Platform to solve possible problems.

1.18.1 1 - Web application not showing in the web browser

In case the web application does not show in the web browser, do the following steps.

1. **Check the internet connection:** Verify if the user has internet connection since it is required to interact with the INNUENDO Platform.
2. **Verify if the frontend service is up:** Check if the frontend server is up by entering the machine with the frontend application. If the service is not running, start it.
3. **Check if Nginx service is up:** Check if the Nginx service is running by typing *service nginx status*. If is not running, start it by typing *service nginx start*.
4. **Check the Nginx configuration file:** If the above step does not work, check the Nginx configuration file for possible errors.

In case the above steps don't solve the problem, please contact the developer.

1.18.2 2 - Job submission stuck on waiting animation

In case the loading screen does not disappear after submitting jobs to the server, do the following steps:

1. **Verify if all services are up:** Normally this event can be caused by miss-communication between the frontend application and the process controller. Enter the machine running the process controller and check if the service is running. If not, start it.
2. **Check if jobs were submitted:** Enter the user project and go to the *Information* section. Check for the Nextflow Logs and Flowcraft Build Logs. You can also check the submitted jobs by entering the machine running the **process controller** and type *squeue* to check the jobs running.
3. **Re-run if no jobs were submitted:** Remove all the workflows applied to the strains with problems, apply again and re-run the jobs.

In case the above steps don't solve the problem, please contact the developer.

1.18.3 3- Nextflow aborts a pipeline

Enter the project with with the problematic strains and check the Nextflow Logs. In case the pipeline being aborted, resubmit it by clicking on the Retry button which appears below the log. After the submission, refresh the log tab to verify if it is running.

In case the above steps don't solve the problem, contact the developer.

1.18.4 4- Profile classification not showing after chewBBACA run

This can happen because the worker service used for classification and PHYLOViZ Online submission not being running. Enter the machine with the frontend server installed and check if the *worker.py* process is running. If not, start the process.

1.18.5 5- PHYLOViZ Online submission not working

This can happen because the worker service used for classification and PHYLOViZ Online submission not being running. Enter the machine with the frontend server installed and check if the *worker.py* process is running. If not, start the process.

1.18.6 6- PHYLOViZ Online application not showing

Check if the PHYLOViZ Online services are running. To do that, go to the machine were PHYLOViZ Online is installed, go to its source code folder and type *pm2 list*. If all services (app.js and queue_worker.js) are not running, launch them by typing:

```
# For the app.js
pm2 start app.js

# For the queue_worker.js
pm2 start queue_worker.js
```

You can always change the memory and cpus allocated to the processes by running:

```
# Restart app.js with 2 cpu allocated and 8GB of memory
pm2 restart app.js -i 2 --node-args="--max-old-space-size=8192"

# Restart queue_worker.js with 2 cpu allocated and 8GB of memory
pm2 restart queue_worker.js -i 2 --node-args="--max-old-space-size=8192"
```

1.19 REST API

Information on the documented REST API of the INNUENDO REST API and INNUENDO PROCESS CONTROLLER can be found at INNUENDO API wiki.