datarun Documentation

Release 0.2

Camille

November 25, 2016

CONTENTS

Datarun goal is to train and test machine learning models. It is a REST API written in Django.

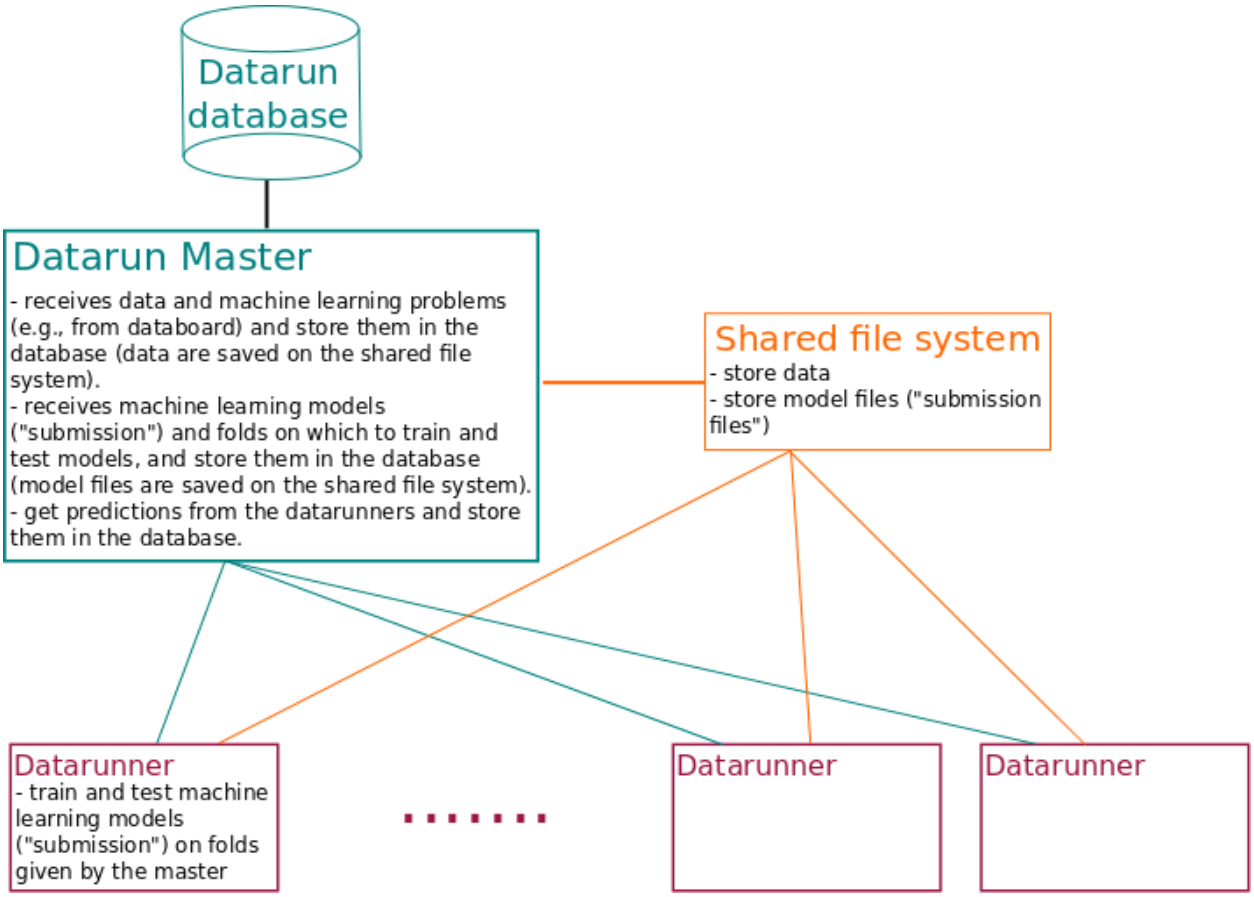
The basic workflow is the following (more details can be found in *How to use datarun?*):

1. Data (on which machine learning models are trained and tested) are sent to datarun.
2. Models and train and test indices of CV fold are send to datarun, which train and test these models on these indices.
3. The resulting predictions can then be requested.

In this documentation, we use the following terminology (which corresponds to the database tables, cf *Models*):

- `RawData` refers to the data on which machine learning models are trained and tested
- `Submission` refers to a machine learning model
- `Submission on cv fold` / `SubmissionFold` refers to a submission and the indices of train and test of a cv fold.

Datarun is made of a master and datarunners, as represented below:



Contents:

CHAPTER

ONE

HOW TO USE DATARUN?

The workflow to use datarun is the following:

1- Send data to datarun

The standard format of a data file excepted by datarun is a csv file whose first row contains the feature and target names, each line corresponds to a data sample.

Here is an example of a standard data file:

```
sepal length,sepal width,petal length,petal width,species
5.1,3.5,1.4,0.2,setosa
4.9,3.0,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
4.6,3.1,1.5,0.2,setosa
```

If your data match the standard data file, you need to send:

- the dataset name (for instance if you use databoard, you can use the problem name)
- your data file
- the name of the target column
- the workflow elements of the problem related to the dataset (for instance feature_extractor, classifier, ...)

If your data do not match the standard data file, you need to send in addition to above:

- a python file with 5 specific functions (an example of such file is `test_files/variable_stars/variable_stars_datarun.py`):
  - `prepare_data(raw_data_path)`
  - `get_train_data(raw_data_path)`
  - `get_test_data(raw_data_path)`
  - `train_submission(module_path, X, y, train_indices)`
  - `test_submission(trained_model, X, test_indices)`
- possibly other data files (if your data are split in different files).

In both cases, to send your data to datarun, you can use:

- a post request to `<master-host>/runapp/rawdata/` (cf *direct requests*, class runapp.views.RawDataList)
- the `post_data` function in the module test_files.post_api (cf *post_api module*)

Note for databoard users:

1. To prepare data for datarun:

- If they match the standard data file, there is nothing to do.
- If they do not match the standard data file,

    1. **create a file `problems/<problem_name>_datarun.py` which corresponds to the above mentionned python file.** Functions `prepare_data(raw_data_path)`, `get_train_data(raw_data_path)`, and `get_test_data(raw_data_path)` are almost exact copies of the same functions defined in `databoard/specific/problems/<problem_name>.py`, except the dependence on `raw_data_path` (which allows datarun to find the data file where it saves it). Be careful to remove all dependencies with `databoard` module. Functions `train_submission(module_path, X, y, train_indices)` and `test_submission(trained_model, X, test_indices)` are exact copies of the same functions defined in the problem workflow (`databoard/specific/workflows/<workflow_name>.py`).
    2. Add in `databoard/specific/problems/<problem_name>.py` a line specifying the above mentionned python file and possible other data files. E.g, `extra_files = extra_files = [vf_raw_filename, os.path.join(problems_path, problem_name, 'variable_star_datarun.py')]` (for the variable stars problem).

2. **To send data to datarun and to split data into train and test dataset, you can use the function `send_data_datarun` of `d`** This function can be called with fab: `fab send_data_datarun:<problem_name>,<datarun_master_url>,`

2- Split data into train and test dataset

If your data match the standard format, you can use:

- a post request to `<master-host>/runapp/rawdata/split/` (cf *direct requests*, class runapp.views.SplitTrainTest)
- the `post_split` function in the module test_files.post_api (cf *post_api module*)

If your data do not match the standard format, you can use:

- a post request to `<master-host>/runapp/rawdata/customsplit/` (cf *direct requests*, class runapp.views.CustomSplitTrainTest)
- the `custom_post_split` function in the module test_files.post_api (cf *post_api module*)

Note for databoard users: To send data to datarun and to split data into train and test dataset, you can use the function `send_data_datarun` of `databoard/db_tools.py`, which uses the functions `post_data` and `post_split` (or `custom_post_split`) of the module test_files.post_api of datarun (cf previous section).

This function can be called with fab: `fab send_data_datarun:<problem_name>,<datarun_master_url>,<dat`

3- Send submission on cv fold to be trained on datarun

To send a submission on cv fold, you can use:

- a post request to `<master-host>/runapp/submissionfold/` (cf *direct requests*, class runapp.views.SubmissionFoldList)
- the `post_submission_fold` function in the module test_files.post_api (cf *post_api module*)

If the associted submission files have already been sent, you'll need to send:

- the id of the associated submission
- the id of the submission on cv fold
- the train and test indices of the cv fold. * after compression (with zlib) and base64-encoding if you use a post request * the raw indices if you use the `post_submission_fold` function
- the priority level (`L` for low or `H` for high) of training this submission on cv fold.
- an indication that you want to force retraining the submission on cv fold even if it already exists (`force="submission_fold"` instead of `force=None`).

If the associated submission files have not been sent, you need to add:

- the id of the associated data. This id can be retrieved using:
- a post request to `<master-host>/runapp/rawdata/` (cf *direct requests*, class runapp.views.RawDataList)
- the `get_raw_data` function in the module test_files.post_api (cf *post_api module*)
- the list of submission files
- an indication that you want to force resending the submission even if its id already exists (`force="submission"` instead of `force=None`).

Note for databoard users: To send a submission on cv fold, you can use the function `train_test_submissions_datarun` of `databoard/db_tools.py` (which uses functions from the module test_files.post_api of datarun).

This function can be called with fab: `fab train_test_datarun:<data_id_datarun>,<datarun_master_url>,`

The <data_id_datarun> is printed when sending data to datarun, or it can be retrieved as mentionned above.

4- Get back your predictions

If you want to get all predictions that have not been requested, you can use:

- a post request to `<master-host>/runapp/testpredictions/new/` (cf *direct requests*, class runapp.views.GetTestPredictionNew)
- the `get_prediction_new` function in the module test_files.post_api (cf *post_api module*)

If you want to get predictions given a list of submission on cv fold ids, you can use:

- a post request to `<master-host>/runapp/testpredictions/list/` (cf *direct requests*, class runapp.views.GetTestPredictionList)
- the `get_prediction_list` function in the module test_files.post_api (cf *post_api module*)

Note for databoard users: To get back predictions, you can use the function `get_trained_tested_submissions_datarun` of `databoard/db_tools.py` (which uses functions from the module test_files.post_api of datarun).

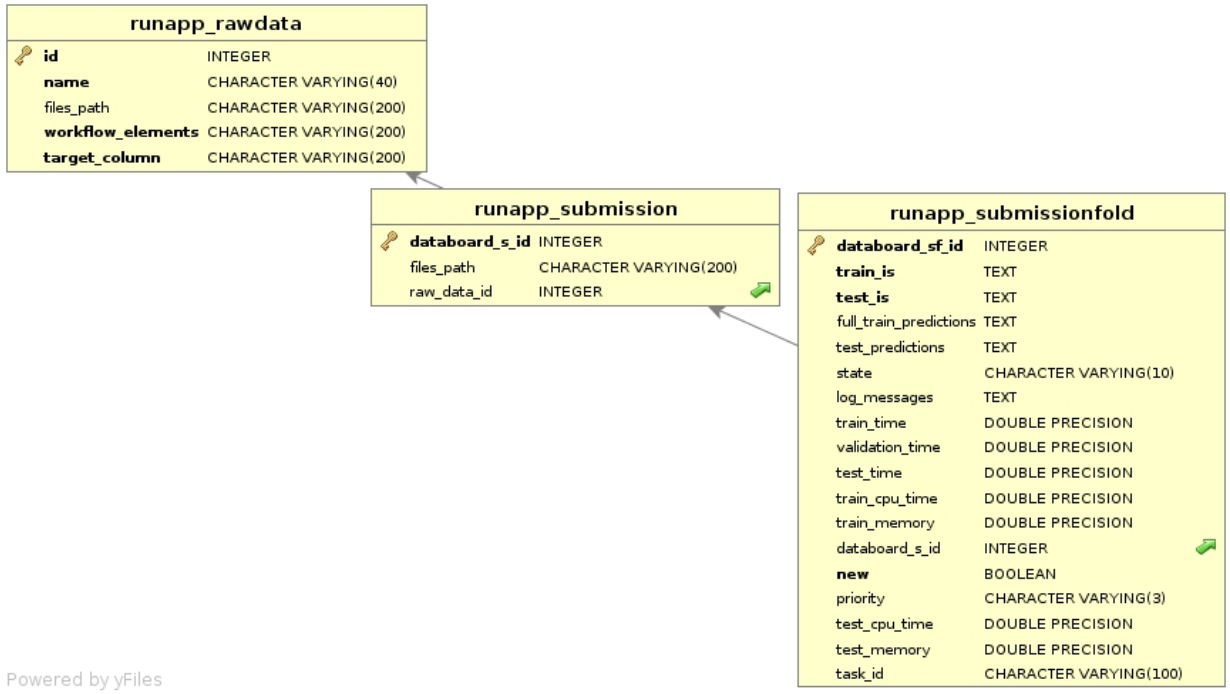
This function can be called with fab: `fab get_trained_tested_datarun:<datarun_master_url>,<datarun_u`

CHAPTER

TWO

MODELS

The database schema is the following:



class runapp.models.**RawData**(**args*, ***kwargs*)

Parameters

- **name** (*string*) – name of the data set
- **files_path** (*string*) – path of file where data are saved
- **workflow_elements** (*string*) – list of workflow elements used to solve the RAMP
- **column** (*target*) – name of the target column

class runapp.models.**Submission**(**args*, ***kwargs*)

Parameters

- **databoard_s_id** (*IntegerField(primary_key=True)*) – id of the submission in the db of databoard

- **files_path** (*CharField(max_length=200, null=True)*) – path of submitted files
- **raw_data** (*ForeignKey(RawData, null=True, blank=True)*) – associated raw data

class runapp.models.**SubmissionFold**(**args*, ***kwargs*)

Parameters

- **databoard_sf_id** (*IntegerField(primary_key=True)*) – id of the submission on cv fold in databoard db
- **databoard_s** (*ForeignKey(Submission, null=True, blank=True)*) – associated submission
- **train_is** (*TextField*) – train indices
- **test_is** (*TextField*) – test indices
- **priority** (*CharField, choices.*) – priority to train-test the fold ('L' for low priority, 'H' for high priority)
- **full_train_predictions** (*TextField*) – predictions of the entire train dataset
- **test_predictions** (*TextField*) – predictions of the test dataset
- **state** (*CharField, choices.*) – TODO, TRAINED, VALIDATED, TESTED, ERROR
- **log_messages** (*TextField*) – logs recorded during train and test
- **train_time** (*FloatField, default=0.*) – real clock training time
- **validation_time** (*FloatField, default=0.*) – real clock validation time
- **test_time** (*FloatField, default=0.*) – real clock testing time
- **train_cpu_time** (*FloatField, default=0.*) – training cpu time
- **train_memory** – peak memory usage during train and test (in kb)
- **test_cpu_time** – test cpu time
- **test_memory** (*FloatField, default=0.*) – peak memory usage durnig train and test (in kb)
- **new** (*BooleanField, default=True.*) – True when it has not already been sent by the API

CHAPTER

THREE

REQUESTS

You can either make direct requests to the datarun API, or use the post_api function.

direct requests

post_api module

test_files.post_api.**custom_post_split**(*host_url*, *username*, *password*, *raw_data_id*)

To split data between train and test on datarun using a specific prepare_data function sent by databoard

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication
- **raw_data_id** (*integer*) – id of the raw dataset on datarun

test_files.post_api.**get_prediction_list**(*host_url*, *username*, *password*, *list_submission_fold_id*)

Get predictions given a list of submission on cv fold ids

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication
- **list_submission_fold_id** (*list*) – list of submission on cv fold ids from which we want the predictions

test_files.post_api.**get_prediction_new**(*host_url*, *username*, *password*, *raw_data_id*)

Get all new predictions given a raw data id

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication
- **raw_data_id** (*integer*) – id of a data set from which we want new predictions

test_files.post_api.**get_raw_data**(*host_url*, *username*, *password*)
Get all raw data sets

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication

test_files.post_api.**get_submission_fold**(*host_url*, *username*, *password*)
Get all submission on cv fold (all attributes)

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication

test_files.post_api.**get_submission_fold_detail**(*host_url*, *username*, *password*, *submission_fold_id*)
Get details about a submission on cv fold given its id

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication
- **submission_fold_id** – id of the submission on cv fold
- **submission_fold_id** – integer

test_files.post_api.**get_submission_fold_light**(*host_url*, *username*, *password*)
Get all submissions on cv fold only main info: id, associated submission id, state, and new

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication

test_files.post_api.**post_data**(*host_url*, *username*, *password*, *data_name*, *target_column*, *workflow_elements*, *data_file*, *extra_files=None*)
To post data to the datarun api. Data are compressed (with zlib) and base64-encoded before being posted.

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication
- **data_name** (*string*) – name of the raw dataset
- **target_column** (*string*) – name of the target column
- **workflow_elements** (*string*) – workflow elements associated with this dataset, e.g., feature_extractor, classifier
- **data_file** (*string*) – name with absolute path of the dataset file

- **extra_files** (*list of string*) – list of names with absolute path of extra files (such as a specific.py)

test_files.post_api.**post_split**(*host_url*, *username*, *password*, *held_out_test*, *raw_data_id*, *random_state=42*)

To split data between train and test on datarun

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication
- **held_out_test** (*float (between 0 and 1)*) – ratio of data for the test set
- **raw_data_id** (*integer*) – id of the raw dataset on datarun
- **random_state** (*integer*) – random state to be used in the shuffle split

test_files.post_api.**post_submission_fold**(*host_url*, *username*, *password*, *sub_id*, *sub_fold_id*, *train_is*, *test_is*, *priority='L'*, *raw_data_id=None*, *list_submission_files=None*, *force=None*)

To post submission on cv fold and submission (if not already posted). Submission files are compressed (with zlib) and base64-encoded before being posted.

Parameters

- **host_url** (*string*) – api host url, such as http://127.0.0.1:8000/ (localhost)
- **username** (*string*) – username to be used for authentication
- **password** (*string*) – password to be used for authentication
- **sub_id** (*integer*) – id of the submission on databoard
- **sub_fold_id** (*integer*) – id of the submission on cv fold on databoard
- **train_is** (*numpy array*) – train indices for the cv fold
- **test_is** (*numpy array*) – test indices for the cv fold
- **priority** (*string*) – priority level to train test the model: L for low and H for high
- **raw_data_id** (*integer*) – id of the associated data, when submitting a submission
- **list_submission_files** (*list*) – list of files of the submission, when submitting a submission
- **force** (*string*) – to force the submission even if ids already exist force can be 'submission, submission_fold' to resubmit both or 'submission, submission_fold' to resubmit only the submission on cv fold. None by default.

- *How to run it locally?*
- *How to run it on stratuslab openstack?*

CHAPTER

FOUR

HOW TO RUN IT LOCALLY?

1. Install the application

Clone the project: `git clone https://github.com/camillemarini/datarun.git`

Install dependencies (might be useful to create a virtual environment before, eg using virtualenv and virtualenvwrapper):

1. For numpy, scipy, and pandas (for Unbuntu & Debian users): `sudo apt-get install python-numpy python-scipy python-pandas`
2. `pip install -r requirements.txt`

Install RabbitMQ (celery broker): `sudo apt-get install rabbitmq-server`

Install Redis and set it up for our app (celery result backend):

:: sudo apt-get install -y redis-server pip install redis sudo sed -i "331a requirepass $DR_DATABASE_PASSWORD" /etc/redis/redis.conf sudo service redis-server restart

2. Set up the database

Datarun uses a Postgres database. Before starting, install postgres if needed and create a database with `createdb database_name`.

3. Define environment variables

- `DR_WORKING_ENV`: `PROD` for production environment or `DEV` for development env
- `DIR_DATA`: directory where to save data
- `DIR_SUBMISSION`: directory where to save submissions
- `DR_DATABASE_NAME`: database name
- `DR_DATABASE_USER`: database user name
- `DR_DATABASE_PASSWORD`: database user password (do not use special characters)
- `DR_EMAIL`: email for the platform superuser
- `CELERY_SCHEDULER_PERIOD`: period (in min) at which the scheduler checks new trained models and saves them in the database. Ex: `*/2` for every 2 min.
- `RMQ_VHOST`: RabbitMQ vhost name

- `IP_MASTER`: ip address of the master, here: `localhost`

If your are using virtualenvwrapper, you can store these variables in `$VIRTUAL_ENV/bin/postactivate`

4. Apply migrations

Run: `python manage.py migrate`

5. Create a superuser

Run: `python manage.py createsuperuser`

6. Run the server (localhost)

Run: `python manage.py runserver`

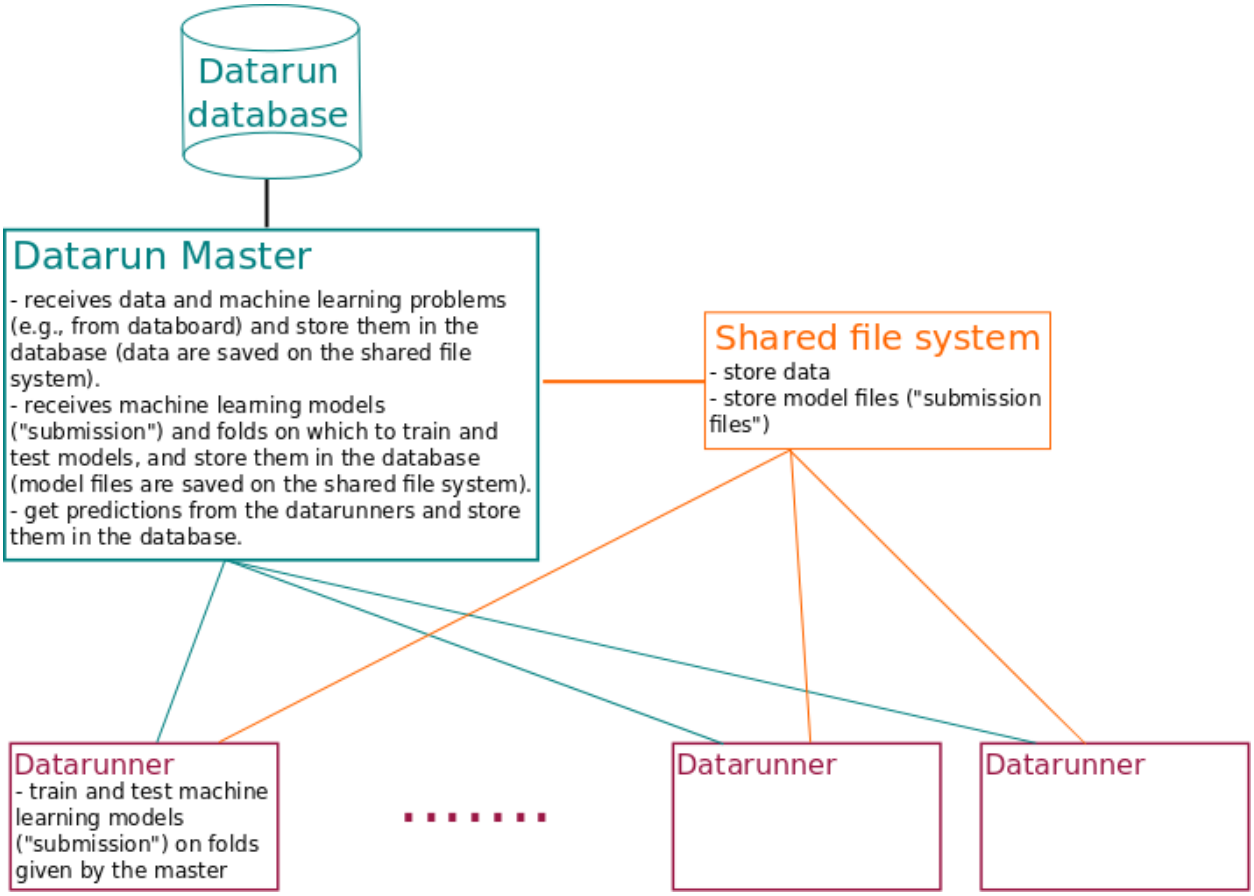
7. Start celery worker and scheduler

Run: `bash test_files/cmd_workers.sh start 2 1` for 3 workers, of which one is for the scheduler
Note: to start one worker, run: `celery -A datarun worker -l info`

CHAPTER

FIVE

HOW TO RUN IT ON STRATUSLAB OPENSTACK?



There are two possibilities:

1. from scratch using an Ubuntu 14.04 image on openstack, or on any other cloud.
2. using images `datarun_master` and `datarun_runner` on openstack

Note: in both cases, you need a scienceFS account. On your scienceFS disk, create in the root directory a folder called `datarun`.

A. Using an Ubuntu 14.04 image

A1. Start one instance for the master and as many instances as you want for the runners.

Use Ubuntu v14.04 images. For the master, an VM os.2 is enough.

A2. Go to the `script_install` directory and stay there while configuring the master and runners.

A3. Configure the master

- On your local computer, create a file called `env.sh` (do not change this name) with the content below. Do not forget to change the values and be careful **not to commit this file** :-) And **do not add comments to the file**. Make sure that the directory `SCIENCEFS_DATARUN` has been created on the sciencefs disk beforehand.

```
export SCIENCEFS_LOGIN='login_for_scienceFS_account'
export SCIENCEFS_DATARUN='path_of_sciencefs_disk'
export DR_DATABASE_NAME='database_name'
export DR_DATABASE_USER='database_user'
export DR_DATABASE_PASSWORD='database_password'
export DIR_DATA='/mnt/datarun/data'
export DIR_SUBMISSION='/mnt/datarun/submission'
export USER_LOGIN='user_name'
export USER_PSWD='user_password'
export CELERY_SCHEDULER_PERIOD='*/2'
export DR_EMAIL='mail@emailworld.com'
export RMQ_VHOST='rabbitMQ_vhost_name'
export IP_MASTER=$(/sbin/ifconfig eth0 | grep "inet addr" | awk -F: '{print $2}' | awk '{print $
```

- Run:

```
bash scp_master_stratuslab.sh master_address scienceFS_private_key
```

  with `master_address` being the master server address (e.g., `onevm-81.lal.in2p3.fr`) and `scienceFS_private_key` being the file name (with absolute path) of the private key to connect to ScienceFS account. This will scp to the master some files that are needed to configure the master.

- Ssh to the instance and run:

```
bash deploy_master_stratuslab.sh
source ~/.bashrc
```

- Once you've checked that the app is running (going to <master_address>/admin for instance), do not forget to change the Django setting `DEBUG` to False and add the server name (<IP_MASTER>) in `ALLOWED_HOSTS` (preceded with a dot). In `/home/datarun/datarun/settings.py`:

```
DEBUG = False
ALLOWED_HOSTS = ['.<IP_MASTER>']
```

A4. Configure runners

- On your local computer in the folder `script_install`, create a file called `env_runner.sh` (be careful to use the name `env_runner.sh`) with the content below. Do not forget to change the values and

be careful not to commit this file :-) And **do not add comments to the file**. Make sure that the directory `SCIENCEFS_DATARUN` has been created on the sciencefs disk beforehand.

```
export SCIENCEFS_LOGIN='login_for_scienceFS_account'
export SCIENCEFS_DATARUN='path_of_sciencefs_disk'
export DR_DATABASE_USER='database_name'
export DR_DATABASE_PASSWORD='database_password'
export DIR_DATA='/mnt/datarun/data'
export DIR_SUBMISSION='/mnt/datarun/submission'
export RMQ_VHOST='rabbitMQ_vhost_name'
export IP_MASTER='xxx.yyy.zz.aaa'
```

Values of these environment variables must be the same as what you defined in `env.sh`, they are used to connect to the master and read data from it.

- On your local computer, create a file `list_runners.txt` containing the list of runners address address, the number of tasks you want to be run concurrently on each runner, the list of queues processed by the workers (at least one of each among `L`, `H`, `celery`), and the hard and soft time limit in seconds:

```
address_runner_1 number_task_runner_1 list_queues_1 hard_time_limit_1 soft_time_limit_1
address_runner_2 number_task_runner_2 list_queues_2 hard_time_limit_2 soft_time_limit_2
...
address_runner_3 number_task_runner_3 list_queues_3 hard_time_limit_3 soft_time_limit_3
```

Example:

```
134.158.75.112 2 L,celery 360 300
134.158.75.113 3 H 240 200
```

- Run:

```
bash scp_runner_stratuslab.sh list_runners.txt scienceFS_private_key
```

As above, `scienceFS_private_key` is the file name (with absolute path) of the private key to connect to ScienceFS account. This will scp some files to the runners and configure them (by executing the script `deploy_runner_stratuslab.sh`)

You should now be ready to use datarun on stratuslab!

B. Using images `datarun_master` and `datarun_runner` on openstack

B1. Start one instance for the master and as many instances as you want for the runners.

Use the image `datarun_master` for the master and `datarun_runner` for runners.

B2. Go to the `script_install` directory and stay there while configuring the master and runners.

B3. Configure master

1. Ssh to the instance
2. Go to `/home/datarun/script_install`

3. Run `bash deploy_master_from_image.sh`

B4. Configure runners

- On your local computer, create a file `list_runners.txt` containing the list of runners address address, the number of taskss you want be run concurrently on each runner, the list of queues processed by the workers (at least one of each among `L`, `H`, `celery`), and the hard and soft time limit in seconds:

```
address_runner_1 number_task_runner_1 list_queues_1 hard_time_limit_1 soft_time_limit_1
address_runner_2 number_task_runner_2 list_queues_2 hard_time_limit_2 soft_time_limit_2
...
address_runner_3 number_task_runner_3 list_queues_3 hard_time_limit_3 soft_time_limit_3
```

  Example:

```
134.158.75.112 2 L,celery 360 300
134.158.75.113 3 H 240 200
```

- Run:

  :: bash scp_runner_from_image.sh list_runners.txt

  This will configure the runners (by executing the script `deploy_runner_from_image.sh`). **Check that the sciencefs disk has been correclty mounted** (ssh to the instance and check if `/mnt/datarun` is not empty), sometimes it fails...

C. How to install missing packages on runners

If the package can be installed with pip, run: `runner_install {list_runner.txt} {package name}` with `{list_runner.txt}` being the file mentionned (A and B sections) specifying runners. It is going to run on each runner the following `pip install {package_name}`

CHAPTER

SIX

TESTS

for django tests

Run: `python manage.py test`

for local celery tests

Run in one terminal:

```
cd test_files; bash local_test1.sh
```

Run in another terminal:

```
cd test_files; bash local_test2.sh -d
```

It creates a database, start celery workers, and run tests from `test_files/test_workflow.py` file.

If two 'Oh yeah' are printed, tests are ok!

for tests on stratuslab

Run:

```
cd test_files
bash stratuslab_test.sh master_address username password
```

with `master_address` being the master server address (e.g., `onevm-81.lal.in2p3.fr`), `username` being a datarun user, and `password` its password.

It runs tests from `test_files/test_workflow.py` file.

If two 'Oh yeah' are printed, tests are ok!

CHAPTER

SEVEN

HOW TO DEAL WITH MIGRATIONS?

If you modify the app models, you'll need to migrate the database.

1. modify `runapp/models.py`
2. run `python manage.py makemigrations` which will create a migrations file in `runapp/migrations/`
3. apply the migration with `python manage.py migrate`
4. add the migrations file to your git and commit both the modified `runapp/models.py` and migrations file (so that other contributors can have the migration history)

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

r

t

C

G

P

R

S

T