Cluster access tutorial

To access the cluster and open the links provided in the tutorial you need to be connected to the EPFL wifi or EPFL VPN. Links in the tutorial are provided for you to be able to refer to original sources if you want more details about the steps or if something is not working. However, you should be able to set up the job submission by simply following the instructions in the tutorial. Every new `bash command` in the tutorial is separated by an empty line (if no empty line is present the command is continued to the next line).

The code shown in `this font` is meant to be run in a terminal. If the code appears across multiple lines without any blank lines in between, treat it as a **single command**. A **blank line** between code blocks means you should run the next part as a **separate command**.

Links to the documentation, where you can read information additional to the tutorial

RCP Wiki: https://wiki.rcp.epfl.ch/en/home/CaaS
Run:ai: https://docs.run.ai/v2.18/Researcher/cli-reference/Introduction/
Getting familiar with shell tools: https://missing.csail.mit.edu/

Login to RunAI

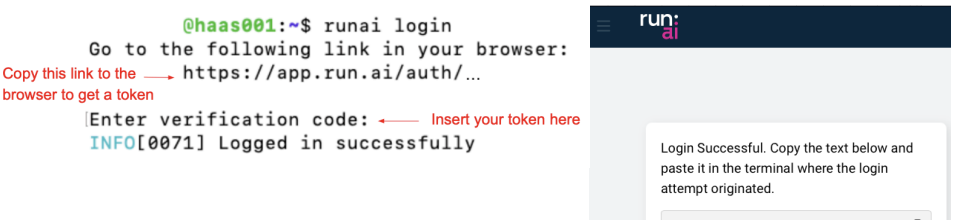
RCP documentation: https://wiki.rcp.epfl.ch/home/CaaS/FAQ/how-to-use-runai

Steps in the following section have to be done only ones - the first time you connect to the cluster. You don't need to repeat them every time you submit a job.

1) SSH to the jumphost. Execute the following command in the command line:
   `ssh <username>@jumphost.rcp.epfl.ch`
   Replace `<username>` with your gaspar username, and enter your password.

2) Login to the RunAI system.
   a) On the server run the command:
      `runai login`
   b) Follow the link outputted by the previous command in the terminal and go to the website. (If asked to log in, press "Continue with SSO")

   c) Copy the text that appears in the special window and insert it into the command line after the `Enter verification code:`



3) Run the next command:
`runai config project course-ee-559-${USER}`
In this command keep `${USER}` as it is, since this is the system variable, and it will be pulled automatically.

Now you can run jobs!

How to run a job

RCP documentation: https://wiki.rcp.epfl.ch/home/CaaS/FAQ/how-to-submit-job-demo

1) From your local computer copy the project files to the home directory on the server (jumphost). To do that in the terminal on your local computer, go to the directory where the `practice_3_repository` is located and run the following command:
`scp -r practice_3_repository <username>@jumphost.rcp.epfl.ch:/home/<username>/`
Replace `<username>` with your Gaspar username.

The command `scp -r` copies the directory from your local pc to the server. Simply it works like `scp -r <path-to-the-directory-on-your-local-pc> <server-address>:<path-to-the-home-directory-on-server>`. You can familiarise yourself with this and other bash commands here https://missing.csail.mit.edu/ and read more about `scp -r` here https://www.geeksforgeeks.org/scp-command-in-linux-with-examples/ .

Similarly, go to the directory where `fruit_dataset` is located and copy it to the home directory on the server too:
`scp -r fruit_dataset <username>@jumphost.rcp.epfl.ch:/home/<username>/`

2) On the jumphost (`ssh <username>@jumphost.rcp.epfl.ch`) submit the job through the runAI.

To submit a job you need to execute the command `runai submit <parameters>` where `<parameters>` should include the details on the type of job you want to run.

For the parameters you must use the next ones:
`--run-as-user` Run in the context of the current user rather than the root user. Only necessary if you want to use generic docker images (e.g. ubuntu, nvcr.io/nvidia/pytorch:25.03-py3, nvcr.io/nvidia/ai-workbench/pytorch:1.0.6) and ensure that you have the correct parameters that specify your user information (UID/GID) to be able to access the persistent storage. However, it is not recommended if your image already has the correct UID/GID, as it creates a new home folder for the user in the container, and this can have some side effects (for example, if Pip packages are installed by default in the user's home within the image, relocating the home will make them unreachable).
`--pvc <dir>` Mounts the directory `dir` to the run. Basically if you want to access a directory from runAI you need to mount it first.
`--image <docker_image_name>` Specifies what docker image `docker_image_name` should be used when running the job
`--command -- <command_to_run>` Specifies which command should be executed. To run a python file `file.py` the running command should be `python3 file.py`.

`--e <variable>` Set environment variables like HOME in the container

`--gpu <number-of-gpus-need>` Set number of GPUs needed for the job. Always request 1, unless you really need more for your model and your code is adopted for the multi-GPU training.

For example, you can submit a job with a base docker image `nvcr.io/nvidia/ai-workbench/pytorch:1.0.6` provided by nvidia with:

```
runai submit --run-as-user --image nvcr.io/nvidia/ai-workbench/pytorch:1.0.6  --gpu 1 --pvc home:${HOME} -e HOME=${HOME} --command -- python3 ~/practice_3_repository/practice_3_simplified.py --dataset_path ~/ --results_path ~/practice_3_repository/results/
```

In detail this tells the server to execute a command (`python3 ~/practice_3_repository/practice_3_simplified.py --dataset_path ~/ --results_path ~/practice_3_repository/results/`, where parameters `--dataset_path` and `--results_path` specify the directories on the server where the data is going to be loaded from and where the results are going to be saved to) while using (`nvcr.io/nvidia/ai-workbench/pytorch:1.0.6`) pythorch1.0.6 docker image and one gpu (`--gpu 1`), and include the path to the files that you store in the home directory (`--pvc home:${HOME} -e HOME=${HOME}`)

The `practice_3_repository/practice_3_simplified.py` script contains training of the model that was provided in Practice 3 and saving the results. When your job is complete, the trained model checkpoint will appear in the `/home/<username>/practice_3_repository/results/checkpoint/` directory and the .png graphs of the losses will appear in the `/home/<username>/practice_3_repository/results/` directory.

Running an interactive job with a base docker image `nvcr.io/nvidia/ai-workbench/pytorch:1.0.6` provided by nvidia:

```
runai submit --run-as-user --image nvcr.io/nvidia/ai-workbench/pytorch:1.0.6 --gpu 1 --pvc course-ee-559-scratch:/scratch --pvc home:${HOME} -e HOME=${HOME} --interactive --attach
```

3) Other commands to interact with runAI:
   a) Output the logs for the job. Logs contain information that was printed when running the job or errors if the job failed:
      `runai logs <job-name>`
   b) Delete the job
      `runai delete job <job-name> -p course-ee-559-<username>`
   c) List all jobs
      `runai list`
   d) Describe the status of the job
      `runai describe job <job-name> -p course-ee-559-<username>`

How to build your own docker image

https://wiki.rcp.epfl.ch/home/CaaS/FAQ/how-to-build-a-container-part1
https://wiki.rcp.epfl.ch/en/home/CaaS/FAQ/how-to-build-a-container-part2

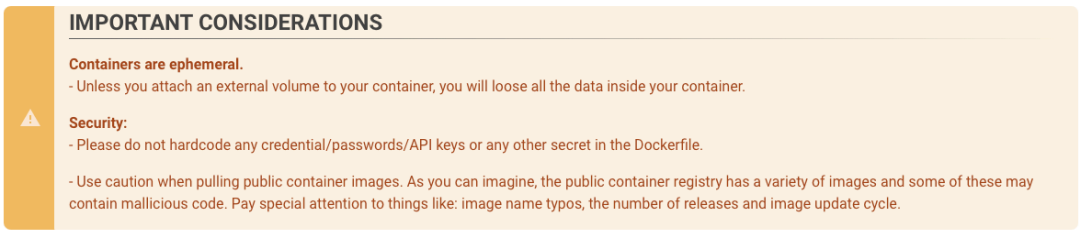
1) **On your local computer** install docker. You can download it here https://www.docker.com/

2) You would need to create a docker image on your **local computer** and for that create a folder where you are going to do that. Create a folder `ee559_docker_env`. In this folder create a `requirements.txt` file, where you will list all the necessary requirements to run your project, and a `Dockerfile`, which will contain all of the instructions for building a docker.

3) In `Dockerfile`:
   a) Take the base Dockerfile we prepared for you (it is provided together with the instructions). Carefully read the file. Snapshot of the file:

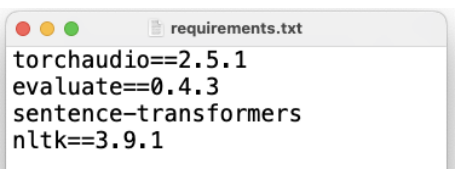
```
###################################
# RCP CaaS requirement (Image)
###################################
# The best practice is to use an image
# with GPU support pre-built by Nvidia.
# https://catalog.ngc.nvidia.com/orgs/nvidia/containers/

# For example, if you want to use an image with pytorch already installed
# FROM nvcr.io/nvidia/pytorch:25.03-py3 or FROM nvcr.io/nvidia/ai-workbench/pytorch:1.0.6
# In this example, we'll use the second image.

FROM nvcr.io/nvidia/ai-workbench/pytorch:1.0.6

###################################
# RCP CaaS requirement (Storage)
###################################
# Create your user inside the container.
# This block is needed to correctly map
# your EPFL user id inside the container.
# Without this mapping, you are not able
# to access files from the external storage.
ARG LDAP_USERNAME
ARG LDAP_UID
ARG LDAP_GROUPNAME
ARG LDAP_GID
RUN groupadd ${LDAP_GROUPNAME} --gid ${LDAP_GID}
RUN useradd -m -s /bin/bash -g ${LDAP_GROUPNAME} -u ${LDAP_UID} ${LDAP_USERNAME}

###################################

# Copy your code inside the container
RUN mkdir -p /home/${LDAP_USERNAME}
COPY ./ /home/${LDAP_USERNAME}

# Set your user as owner of the new copied files
RUN chown -R ${LDAP_USERNAME}:${LDAP_GROUPNAME} /home/${LDAP_USERNAME}

# Install required packages
RUN apt update
RUN apt install python3-pip -y

# Set the working directory in your user's home
WORKDIR /home/${LDAP_USERNAME}
USER ${LDAP_USERNAME}


# Install additional dependencies
RUN pip install -r requirements.txt
```

Keep `LDAP_USERNAME`, `LDAP_UID`, `LDAP_GROUPNAME` and `LDAP_GID` in the file as they are (do not replace with your Gaspar username or anything else). The `ARG USER_UID` instruction in a Dockerfile is used to define a build-time variable. It means that the value of `USER_UID` is provided at build time (step 6). This allows sharing a Dockerfile so each person can build the image with parameters specific to their needs, such as their own `UID/GID`.



   b) Select the base image based on which you want to create your own. You can use the same one we are using in the example: nvcr.io/nvidia/ai-workbench/python-basic:1.0.6. Alternative can be found here https://catalog.ngc.nvidia.com/orgs/nvidia/containers/ (e.g. nvcr.io/nvidia/pytorch:23.11-py3 , nvcr.io/nvidia/ai-workbench/pytorch:1.0.6). You don't have to create a docker image if the default kernel is enough for you.

4) In your `requirements.txt` file write down the python libraries that you will need for your project (like torchaudio, sentence-tranformer) and that are **not included** in the base image on which you are creating your docker.



Do not list the libraries that are already installed in the base docker image in the requirements, since this might create issues with versioning of the already installed libraries. To check which libraries are installed in the base docker image `nvcr.io/nvidia/ai-workbench/python-basic:1.0.6` a) initialise the base docker image `docker run nvcr.io/nvidia/ai-workbench/python-basic:1.0.6` b) check which pip libraries are already installed `pip list`)

5) Find out your LDAP_USERNAME / UID / GROUPNAME etc. To do that ssh to the jumphost as before
   `ssh <username>@jumphost.rcp.epfl.ch`
   and run the command
   `id`
   Example:

```
$ ssh mfontes@jumphost.rcp.epfl.ch

mfontes@haas001:~$ id
uid=248658(mfontes) gid=11202(RCP-GE-StaffU) groups=11202(RCP-GE-StaffU),20249(VPA-AVP-CP-StaffU), ...
```

   Save the printed parameters somewhere for yourself to use later and exit the ssh connection.

6) On your local computer build docker image: `ee559-mini-project-test`, in terminal go to the run the `ee559_docker_env` folder and run the command changing the `<username>` everywhere to your Gaspar username and `<uid>` to the UID you received during the previous operation:

```
docker build --platform linux/amd64 . --tag
registry.rcp.epfl.ch/ee-559-<username>/my-toolbox:v0.1 \
    --build-arg LDAP_GROUPNAME=rcp-runai-course-ee-559_AppGrpU \
    --build-arg LDAP_GID=84650 \
    --build-arg LDAP_USERNAME=<username> \
    --build-arg LDAP_UID=<uid>
```

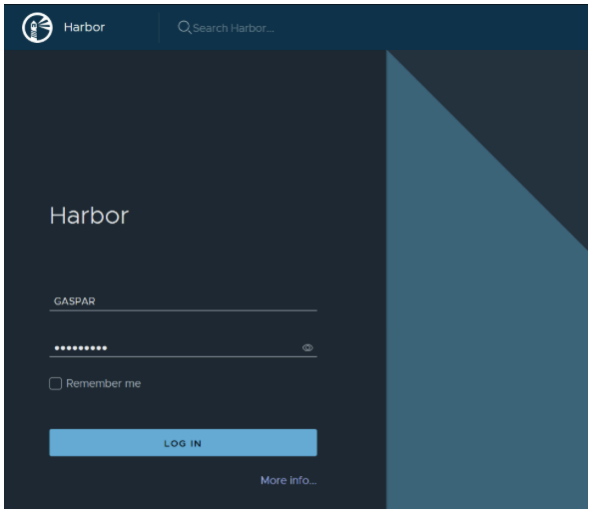
   In the `--tag` parameter `:v0.1` defines the version of your docker image. `registry.rcp.epfl.ch` is needed so that later you can push your docker image to the registry. If later in the project you would need to update the packages, you can rebuild your docker while preserving the name of the project and only changing the version, e.g. to 0.2. `LDAP_GROUPNAME` and `LDAP_GID` are fixed for the course and should not be changed.

7) Check that image is already build:

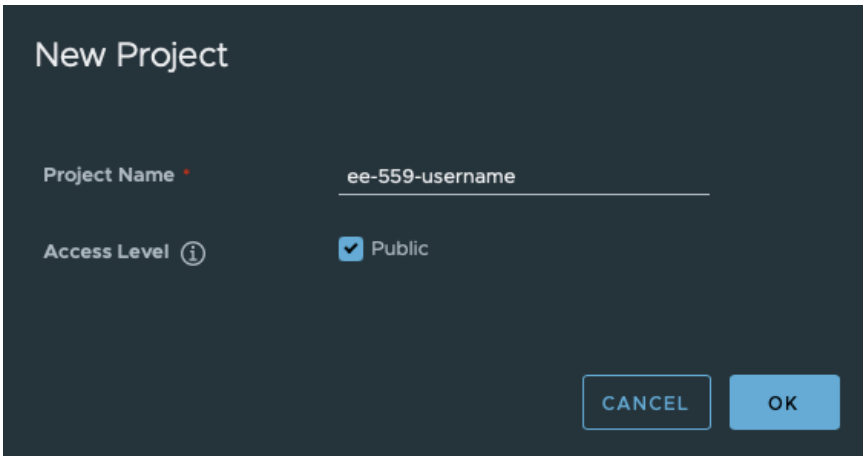
```
docker images
```

```
REPOSITORY                                              TAG       IMAGE ID       CREATED          SIZE
registry.rcp.epfl.ch/ee-559-<username>/my-toolbox        v0.1      127cc75acea5   13 minutes ago   18GB
```

8) Transfer the image to the registry so that you will be able to have access to it on the cluster https://wiki.rcp.epfl.ch/home/CaaS/FAQ/how-to-registry:
   a) Login to https://registry.rcp.epfl.ch with your gaspar username and password



b) Press New project and create a new **PUBLIC** project with the name `ee-559-<username>`. Replace <username> with your Gaspar username. Same as the one you previously built a docker with.



c) Go back to the command line of your PC and log in to the registry with docker. Insert your Gaspar username and password.

```
docker login registry.rcp.epfl.ch
```

```
Username: <username>

i Info → A Personal Access Token (PAT) can be used instead.
         To create a PAT, visit https://app.docker.com/settings


Password:
Login Succeeded
```

d) Push the docker image to the registry with the command

```
docker push registry.rcp.epfl.ch/ee-559-username/my-toolbox:v0.1
```

9) To run the job with your custom container `registry.rcp.epfl.ch/ee-559-username/my-toolbox:v0.1` on server execute `runai submit <parameters>` command with `<parameters>` specified with respect to the type of the job you are submitting.

Submitting a job with your custom docker image:

```
runai submit --run-as-user --image
registry.rcp.epfl.ch/ee-559-username/my-toolbox:v0.1  --gpu 1 --pvc home:${HOME} -e
HOME=${HOME} --command -- python3 ~/practice_3_repository/practice_3_simplified.py
--dataset_path ~/ --results_path ~/practice_3_repository/results/
```

Running an interactive job with a base docker image :

```
runai submit --run-as-user --image
registry.rcp.epfl.ch/ee-559-username/my-toolbox:v0.1  --gpu 1 --pvc home:${HOME} -e
HOME=${HOME} --command -- python3 ~/practice_3_repository/practice_3_simplified.py
--dataset_path ~/ --results_path ~/practice_3_repository/results/
```

Success!

Summary: recipe for running your project

1) Create your own docker image or find a base image that contains all of the necessary libraries.

2) Copy project files and dataset to your home directory (`/home/<username>/`) and shared group directory (`/mnt/course-ee-559/collaborative/group-<group-number>/`, where `<group-number>` should be replaced with the number of your group) on the server.
Note: the shared group directory is going to be created for you soon, see announcement for a notification.

   For the project files if you use github repository, where you can store all the code and collaborate. You can find information about how to create a github repo here https://docs.github.com/en/get-started/start-your-journey/hello-world . Make sure to make your repository is Private. Then you can just clone your repository on the server with `git clone repository_name` and pull updated versions of the code with `git pull`.

   Storing the project in a shared directory is needed so that you can easily exchange files on the server (e.g. if someone from the group already trained the model you can share the weights with your collaborators on the shared directory of your group). Here we have shown how to work with the project taking as an example `practice_3_repository`, which contains multiple Python files that interact with each other.

   If you have never used `scp` command in a command line before, you can read more about it here https://www.geeksforgeeks.org/scp-command-in-linux-with-examples/ .

   To copy these files to the home directory on the server, on your local pc go to the folder where `practice_3_repository` and `fruit_dataset` folders are located and run the `scp -r <path-to-the-directory-on-your-local-pc> <server-address>:<path-to-the-home-directory-on-server>` command:

   ```
   scp -r practice_3_repository
   <username>@jumphost.rcp.epfl.ch:/home/<username>/

   scp -r fruit_dataset <username>@jumphost.rcp.epfl.ch:/home/<username>/
   ```

   To copy these files to the shared directory on the server:

   ```
   scp -r fruit_dataset
   <username>@jumphost.rcp.epfl.ch:/mnt/course-ee-559/collaborative/group-<group
   -number>/

   scp -r fruit_dataset
   <username>@jumphost.rcp.epfl.ch:/mnt/course-ee-559/collaborative/group-<group
   -number>/
   ```

3) Test your script first on an interactive node. Jump to an interactive node specifying the exact docker image that you want to use during the training.
```
runai submit  --image  registry.rcp.epfl.ch/ee-559-<username>/my-toolbox:v0.1
--pvc home:${HOME} -e HOME=${HOME} --interactive -g 1 --attach
```

   Run the Python script on the interactive node to test that your file is running correctly (do it on fewer epochs, do not do the full training on the interactive node since this way you are going to occupy resources).
```
python3 ~/practice_3_repository/practice_3_simplified.py  --dataset_path ~/
--results_path ~/practice_3_repository/results/
```

4) Submit the job:
```
runai submit --image  registry.rcp.epfl.ch/ee-559-<username>/my-toolbox:v0.1 --gpu
1  --pvc home:${HOME} -e HOME=${HOME} --command -- python3
~/practice_3_repository/practice_3_simplified.py  --dataset_path ~/ --results_path
~/practice_3_repository/results/
```

5) If your job failed you can check the logs with
`runai logs <job-name>` and follow the status here https://rcpepfl.run.ai/

Things to be careful about and additional info

1) Make sure that you exactly follow all the instructions! This process contains multiple steps and it is sometimes much harder to debug than to do things from scratch.
2) **Store checkpoints** of your models. When you are running the python file, all printed results will be deleted. You can reuse and adopt the file `practice_3_repository/checkpoint_utils.py` for your project where the storing of model weight and optimiser is implemented.
3) When creating your own docker image, make sure that the docker image you take as a base one is compatible with the versions of the library that you are trying to install. E.g. if you are using the base kernel nvcr.io/nvidia/pytorch:25.03-py3 it will be incompatible with the torchaudio since it is based on python 3.12.3 and torchaudio can only be used with python <= 3.11 (compatibility label can be found here https://pytorch.org/audio/main/installation.html). Therefore, if you need torchaudio for your project you might want to use other base images like nvcr.io/nvidia/ai-workbench/pytorch:1.0.6 and install torchaudio on top of this base image.
4) If a lot of students occupied too many resources some of the jobs might be interrupted, if the server is going to be needed for other EPFL needs. That's again why it is important to save checkpoints of the model,optimiser, scheduler and intermediate results.
5) If you submit a training job requesting one or more GPUs but don't use them for two hours, the job is going to be automatically suspended to free up resources for other users in the cluster. This doesn't prevent you from submitting a new job. If you don't need GPUs, you can submit a job without requesting any. If you need a GPU for live testing, request an interactive job using the "--interactive" flag, as this type of job allows you to keep the GPU allocated without continuous usage.
6) Since you are going to work as a group and on different platforms, try making code easily adaptable to different platforms and repositories. It means **no hardcoding!** E.g. Specify
   `device = torch.device("cuda" if torch.cuda.is_available() else "cpu")`
   Pass all the PATHs as `argparse` arguments (as in the `practice_3_repository/practice_3_simplified.py` file), and do not hardcode any paths, api keys and so on.
7) If you want more details about the command, you can always run `command --help`, e.g. `runai submit --help` , which will output some details about the parameters of the command
8) Error 'can't open file' or 'file not found' might signal that you specified the wrong path to the file. First, check that the path to the file is correct.
9) If you don't want to type the password every time you ssh to the server, you can create an ssh key.