Onboarding process for the KIN'L

The following step-by-step guideline will help any user to access and use our KIN'L server. If you face any problem during this setup process, feel free to contact us at jorge.ruiz@hnee.de

We suppose the existence of a fictitious user named Max Mustermann for the following examples.

Preliminary steps

The next steps describe what the user has to do in order to be able to use our server.

1. **The user must activate a VPN to access the HNEE network.** Follow the instructions at HNEE's VPN/Laufwerke webpage and you also need to contact IT support at support@hnee.de or it-support@hnee.de, providing them the required data. Once ITSZ replies to the user with the details for the configuration of the software WireGuard, it will possible to connect to the HNEE network activating it on the software.
2. **The user must activate an authentication method.** There are many different authentication methods that can be used, we propose the creation of SSH keys to authenticate and access the our server. For example, it is possible to manually generate manually an SSH key for macOS following an online reference, there are similar online references for other operative systems. The SSH protocol uses public key cryptography for authenticating hosts and users, and once generated the keys, the private one will be stored locally on the user's machine, while the public one will be shared to the server for accessing to it.
3. **The user must create an user account on our KIN'L server(s).** The user must contact us at jens.mueller@hnee.de in order to create its account, attaching the public SSH key recently created. We'll describe the account creation only on one of the machines that we call `olive`. In this example, we could create the username max.mustermann, and using the IP of the server that we are currently using 10.1.1.151, it is possible login via SSH from a personal computer terminal using the account max.mustermann@10.1.1.151, but this will be described in detail soon.

Notes about the terminal

We will work typically directly with the terminal or console, which based on UNIX-based OS like MacOS or Linux, but Windows is not so different, and this will look like one of the next 5 cases:

- Local machine

```
max@LocalMachine ~ %
```

Therefore, in our examples, when we use `%` for a command prompt, it should be clear that we are referring to the local machine, the personal computer.

- Remote machine

```
max.mustermann@RemoteMachine:~$
```

Therefore, in our examples, when we use `$` for a command prompt, it should be clear that we are referring to the remote machine, the server.

The next examples won't be available just yet until we install the Anaconda distribution, described in the following section, but we include them anyway for better clarity.

- Python interpreter

```
>>>
```

- Conda environment

```
(base) max.mustermann@RemoteMachine:~$
```

Where on this example we are connected to the remote machine.

- Conda different environment

```
(testenv) max.mustermann@RemoteMachine:~$
```

Where on this example, the new Conda environment has been defined by the user in the remote machine as `testenv`.

Connecting to the server

The following procedure should be done anytime the connection with the server is lost, for example, when the user laptop has been disconnected because of inactivity or when it has been turned off. Sometimes, the WireGuard software would still be connected to the VPN (but

not necessarily to the server), affecting the internet connectivity of the user's computer, so it must be restarted.

1. Activate the VPN connection to the HNEE network via the WireGuard software.
2. Open terminal on the local machine and type

```
max@MaxLaptop ~ % ssh max.mustermann@10.1.1.151
```

and in our example, this will ask for the associated passphrase that was created when the SSH keys were created.

3. Once logged in, the user should see on the terminal something like

```
max.mustermann@olive:~$
```

and be able to read a short message ending with some further documentation to be read in `/usage_notes.md`.

4. It's possible to verify the connection with a command such as

```
max.mustermann@olive:~$ nvidia-smi
```

that should return the information about the Nvidia GPU devices installed in the server.

Most of the instructions that follows will be implemented directly on the server. It is important to address that a Linux distribution is installed on the server, so most of the instructions will take this into consideration.

Setting up the Python environment

Python has established itself as one of the main programming languages for machine learning, and most of the state-of-the-art libraries for data analysis, deep learning (and even local Large Language Models, i.e. LLMs), etc. have been developed for it. Some well known examples are *scikit-learn*, *TensorFlow*, *PyTorch* (and open-source LLM families such as *Llama*), etc. To install Python and their main packages and libraries easily, we recommend the Anaconda distribution, although this is not strictly necessary, it just simplifies many processes. The general process for installing Anaconda in Linux is found on their Installing on Linux website, and the current distribution is typically found on the Anaconda download webpage

1. Download Conda. In particular, from their official webpage and for our Linux server we should download the Linux 64-Bit (x86) Installer.

```
max.mustermann@olive:~$ wget https://repo.anaconda.com/archive/Anaconda3-2024.02-1-Linux-x86_64.sh
```

which is the (Python 3.11) version as of 10.05.2024, but this might be different in the incoming future.

2. Once downloaded, install Conda

```
max.mustermann@olive:~$ bash Anaconda3-2024.02-1-Linux-x86_64.sh
```

and to make the changes take effect, closing and then re-opening the terminal window should suffice.

3. If Conda was installed correctly, from now on the `base` environment will be activated by default every time the terminal is opened, so it should look like

```
(base) max.mustermann@olive:~$
```

where `(base)` is the default environment and it will appear every time, even when the session is initiated or the computer restarted.

4. To further check the Conda installation, we could run for example

```
(base) max.mustermann@olive:~$ python --version
```

or

```
(base) max.mustermann@olive:~$ conda list
```

and a list of installed (Python) packages will appears, if it has been installed correctly.

5. Hello world. We can quickly test some Python code directly on the terminal writing

```
(base) max.mustermann@olive:~$ python
```

and then a basic script such as

```
>>> print("Hallo Welt")
```

where we are now within the Python interpreter. We could try something more sophisticated as

```
>>> [i**2 for i in range(10)]
```

We can exit the Python interpreter and return to the terminal using

```
>>> exit()
```

5. At this stage, the server is even ready do some basic machine learning, and for this we recommend the library scikit-learn -already installed with conda- which has a couple of tutorials, for example, "An introduction to machine learning with scikit-learn" that includes code that could be run directly on the Python interpreter.

Running scripts

One of the most standard ways to run code is using a previously written script (instead of writing code directly on the interpreter), which in the case of Python can be identified by their extension `.py`, and executing it directly on the terminal. To illustrate this we'll download an example that is ready from the scikit-learn library, in this case recognizing hand-written digits, and in particular we'll download the Python script at the end of the webpage called `plot_digits_classification.py`, using the `wget` command as in

```
(base) max.mustermann@olive:~$ wget https://scikit-
learn.org/stable/_downloads/1a55101a8e49ab5d3213dadb31332045/plot_digits_cla
ssification.py
```

This will download the file directly on the user home directory, unless otherwise is specified. To execute the file, we just write

```
(base) max.mustermann@olive:~$ python plot_digits_classification.py
```

and the program will run printing some output on the terminal. It is only necessary that the Python file is in the same folder where we execute it from the terminal, which in this example is the user home directory, as we previously wrote.

Using the Jupyter interfaces

Another emerging trend in the last years for writing and executing (Python) code has been the use of interactive computing tools such as JupyterLab, which is the evolution of Jupyter Notebooks (which in turn is the evolution of IPython). Using JupyterLab can be very useful, not only because their flexible interface is a very powerful interactive development environment for notebooks, code, and data, but also because it can replicate certain basic functions as a graphical user interface of an (Linux) operative system, for users not used to the terminal.

JupyterLab locally

Assuming Conda is installed locally, to run a JupyterLab session locally the user should simply write

```
(base) max@MaxLaptop ~ % jupyter lab
```

and this should open automatically a new tab on the web browser used by default. To properly use it with Python code we can use the same example as before towards the end of recognizing hand-written digits, downloading the notebook called `plot_digits_classification.ipynb` and opening it from the JupyterLab.

JupyterLab on the server

To run a JupyterLab session on the server is slightly more difficult but it will prove to be extremely handy and useful most of the time. It can be done writing

```
(base) max.mustermann@olive:~$ jupyter lab --ip=0.0.0.0
```

and after executing the command we'll read towards the end something like `Or copy and paste one of these URLs: http://olive:8888/lab?token=91074cfa5e895a158dcac0ea0e7c82fef357684fdfbfc08c`, where the token at the end of course it will be different (every time). To actually open a JupyterLab session it is necessary to copy and paste that URL and replace the name of the server with its actual ip or web address, in our example this would result in

```
http://10.1.1.151:8888/lab?
token=91074cfa5e895a158dcac0ea0e7c82fef357684fdfbfc08c
```

When this is introduced in the URL bar, the browser will redirect to a JupyterLab session running on the server. Following the previous examples, we could use the Jupyter notebook described previously, downloading it first to the server using `wget` as

```
(base) max.mustermann@olive:~$ wget https://scikit-
learn.org/stable/_downloads/eb87d6211b2c0a7c2dc460a9e28b1f6a/plot_digits_cla
ssification.ipynb
```

to the user home directory (or to any other folder!). Then, it will be possible to open it from JupyterLab.

Working with some extra libraries on the server

After the previous Python configuration, the user will be ready for using it on the server anytime, this process was meant to be done typically only once. Nevertheless, we might need to use more specialised libraries like PyTorch on the following example (it might even be possible/necessary to create a new conda environment here, but we'll skip this for the next more advanced section).

1. Install PyTorch locally on the server. According to their webpage, the install command for our server and configuration is

```
(base) max.mustermann@olive:~$ conda install pytorch torchvision torchaudio
pytorch-cuda=12.1 -c pytorch -c nvidia
```

2. Once completed, we can check the installation via

```
(base) max.mustermann@olive:~$ conda list torch
```

3. We can run their PyTorch quickstart tutorial. In this example we'll be using the real power of our server, accelerating the operations in the neural network we defined moving it to the GPU (basically, this is done with the Python command `device = ("cuda" if torch.cuda.is_available())`, that can be appreciated slightly different within the tutorial code).