

# PyTorch with GPU in Jupyter Lab using container-based kernel

The easiest way to kick start deep learning is to use our Jupyter Lab feature with container kernel. This article shows how this is achieved using the OAsis web portal.

## Jupyter Lab

It is an exceptional tool for interactive development, particularly in deep learning models. Jupyter Lab offers a user-friendly interface where you can write, test, and debug your code seamlessly. It's an excellent way to enhance your workflow and optimize your development process.

## Kernel

In Jupyter, a kernel is a program that executes code in a specific programming language. Kernels allow Jupyter to support multiple languages, such as Python, R, and Julia, among others. When you open a Jupyter Notebook, you can select which kernel to use, depending on the language you want to use. Once you select a kernel, any code you run in the notebook will be executed by that kernel. This enables you to work with different languages in the same notebook, making Jupyter a versatile and powerful tool for data science and development.

## Container

It is powerful to use containers as kernels in Jupyter. With this approach, there's no need to set up a conda environment or compile specific Python modules for your model. Instead, you can use a container with all the necessary dependencies and libraries. This eliminates the need for manual configuration and streamlines the deployment process. With container as kernel, you can quickly and easily set up your development environment and start your project immediately.

Numerous containers are readily available on the internet. To help you get started, we have pre-downloaded several useful containers in [/pfss/containers](#). This article will utilize the PyTorch container available through the Nvidia GPU Cloud (NGC).

## Launch Jupyter Lab

To begin, you'll need to initiate a Jupyter Lab instance. This can be done easily through the Oasis web portal. Start by selecting "Jobs" in the top-right corner, followed by "Jupyter Lab." From the launcher window, choose a compute node with a GPU that is available. For our purposes, since the model is relatively uncomplicated, we will be using the smallest MIG.

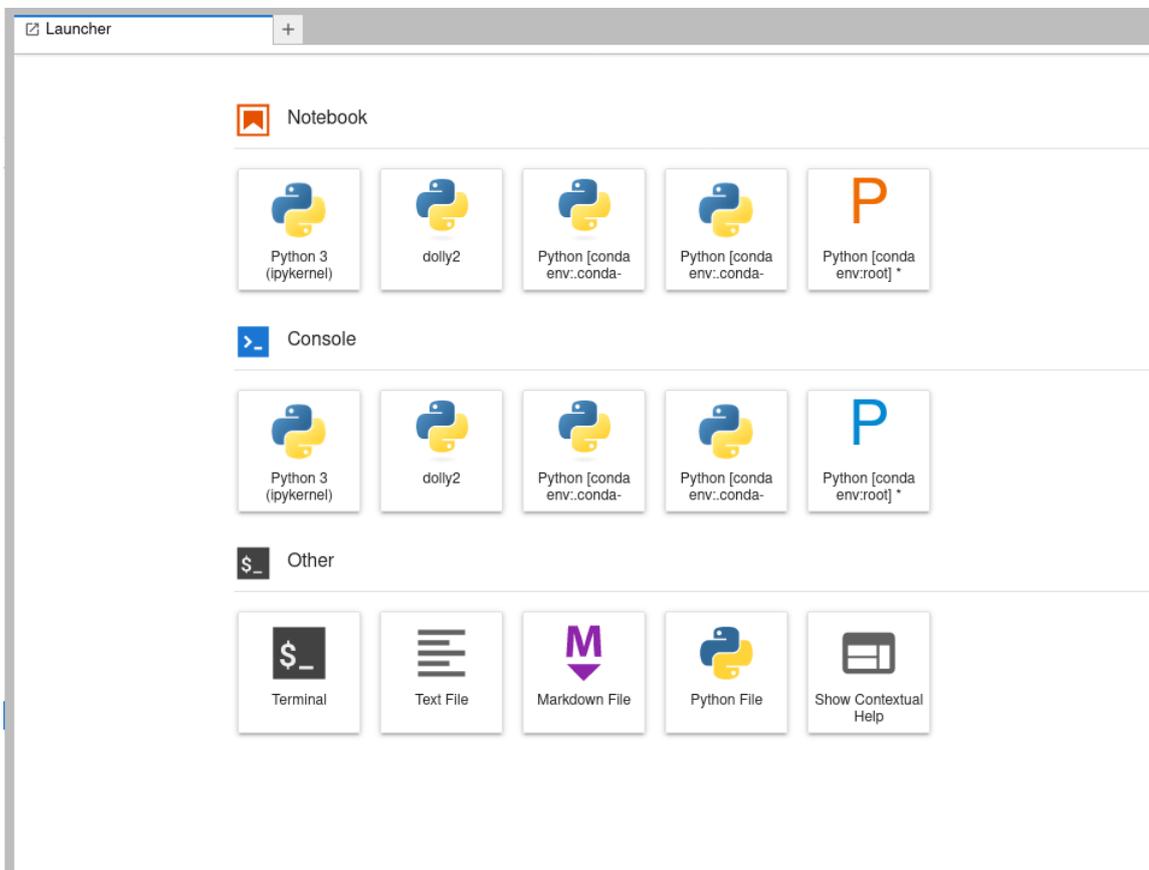
The screenshot shows the Oasis web portal interface. The top navigation bar includes 'HOME', 'ACCOUNTS', 'PARTITIONS', and 'TERMINAL'. The user's account information is visible on the left, including Login ID: loki, Email: loki.ng@app, Posix UID: 2000, and SSH endpoint: ssh.hpccent. The main content area is divided into sections for 'Access Info', 'Invitations', and '7x24 Enquiry Support'. A modal window titled 'Enqueue job' is open, showing the path '/pfs/toolkit/start\_jupyter.sbatch' and a description: 'Launch a Jupyter Lab server, and support accessing from portal. (toggle full description)'. Below this, the 'Node\*' section shows the selected configuration: Partition: gpu, Node: oakibhpcgpu001, CPU: 4 cores, Memory: 8 GB, GPU: 1g.10gb x 1. A 'Select another' link is present. An 'Enqueue now' button is at the bottom right of the modal. A second modal window titled 'Select a node' is also open, showing filters for Partition (gpu), CPU Cores (4), Memory (GB) (8), GPU (1g.10gb), and GPU Count (1). Below the filters is a table of 'Available nodes' with columns for Hostname, State, Cores, Mem, and Gpus. The table shows one node: oakibhpcgpu001, which is in a MIXED state with 24 / 32 (75%) Cores, 451 / 515 GB (88%) Mem, and 1 / 1 a100, 3 / 4 1g.10gb, 1 / 1 3g.40gb GPUs. Numbered callouts 1 through 5 highlight the 'Jobs' link, the 'Enqueue job' modal, the 'Select another' link, the 'Select a node' modal, and the 'Enqueue now' button respectively.

Then you may get access to the launched Jupyter Lab instance by clicking the link in the running jobs window.

The screenshot shows the Oasis web portal interface with the 'Running Jobs' window open. The window title is 'Running Jobs' and it has a filter set to 'Running | Queuing | Completed'. A 'refresh' button is in the top right. Below the filter is a table of running jobs with columns for Job Id, Account, Partition, Duration, Memory, Cpu, and Gpu. The table shows one job: Job Id 88458, Account appcara, Partition gpu, Duration 27s, Memory 0.1 / 7.8 GB (1.46%), Cpu 3s / 1m48s (2.78%), and Gpu 1g.10gb x 1. A 'Jupyter' link is next to the job ID, and a 'Cancel' link is next to the account name. A numbered callout 6 highlights the 'Jupyter' link.

## Setup the container kernel

Our next step is to establish a new folder and generate a kernel.json file for our recently created kernel. To begin, we can launch the terminal from the launcher.

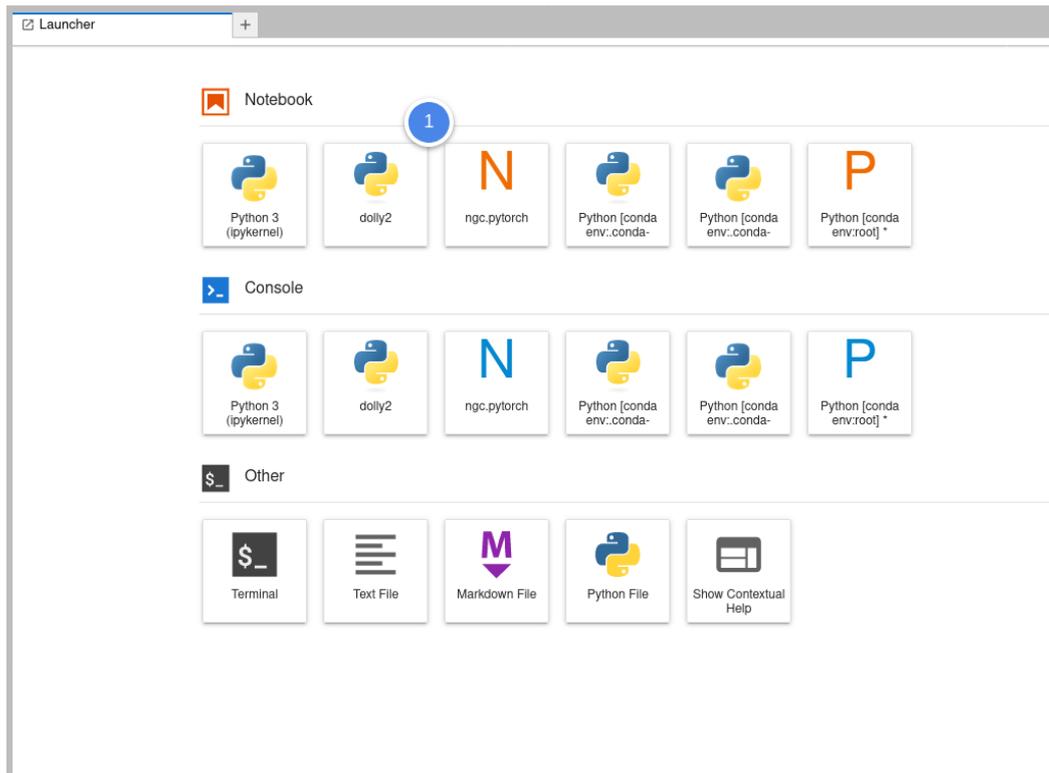


Create the folder with a kernel.json file inside.

```
mkdir -p .local/share/jupyter/kernels/ngc.pytorch

echo '
{
  "language": "python",
  "argv": ["/usr/bin/singularity",
    "exec",
    "--nv",
    "-B",
    "/run/user:/run/user",
    "/pfss/containers/ngc.pytorch.22.09.sif",
    "python",
    "-m",
    "ipykernel",
    "-f",
    "{connection_file}"
  ],
  "display_name": "ngc.pytorch"
}
' > .local/share/jupyter/kernels/ngc.pytorch/kernel.json
```

After completing the prior step, refresh Jupyter Lab. You will then notice the inclusion of a new kernel available in the launcher.



## Notebook

With Jupyter Notebook, users can easily combine code, text, and visual elements into a single document, which can be shared and collaborated on with others.

Now copy the notebook we will use in this article and open it with our new kernel.

```
# go back to the terminal and copy our example notebook
cp /pfss/toolkit/pytorch101.ipynb ./
```

The screenshot shows a JupyterLab notebook with three tabs: '4-training.ipynb', '3-convert-data.ipynb', and 'pytorch101.ipynb'. The 'pytorch101.ipynb' tab is active, and a 'Select Kernel' dialog box is open over it. The dialog box has a title 'Select Kernel' and a subtitle 'Select kernel for: "pytorch101.ipynb"'. A dropdown menu shows 'ngc.pytorch' selected. There are 'Cancel' and 'Select' buttons. The notebook code is as follows:

```
[3]: import torch
import torch.nn as nn

[4]: print(torch.cuda.is_available())
print(torch.cuda.device_count())
print(torch.cuda.get_device_name())

True
1
NVIDIA A100 80GB PCIe MIG 1g.10gb

[5]: class PolynomialEstimator:
    def __init__(self, input_dim, hidden_dim, output_dim):
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.output_dim = output_dim

[6]: # Check if a GPU is available and set the device accordingly
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)

# Initialize the model and move it to the device
model = PolynomialEstimator(input_dim=1, hidden_dim=10, output_dim=1).to(device)

cuda

[7]: import numpy as np

# Define an arbitrary polynomial function
def polynomial_func(x):
    return 2 * x**3 - 4 * x**2 + 7 * x - 3
```

The status bar at the bottom shows 'Simple', '0', '4', 'ngc.pytorch | Idle', 'Mode: Command', 'Ln 1, Col 1', and 'pytorch101.ipynb'. The kernel 'ngc.pytorch' is also visible in the top right corner of the notebook interface.

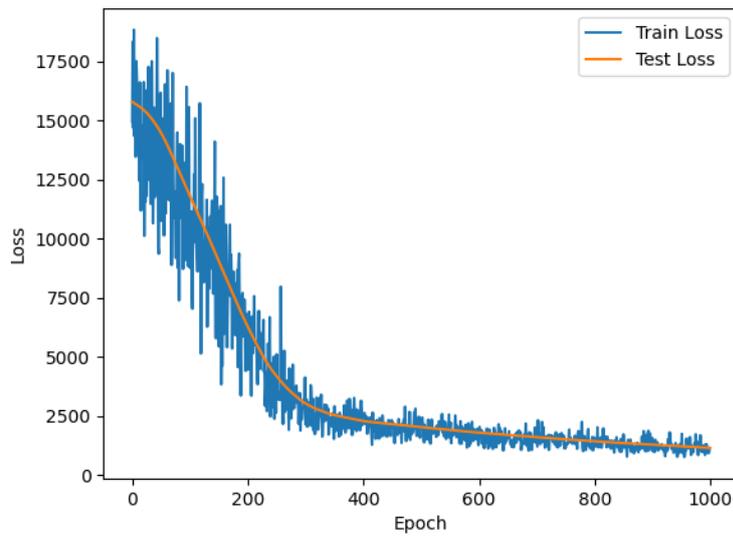
Once you have activated the kernel, you can execute each cell of the notebook and observe the training results, which should appear as shown below.

```
test_losses.append(test_loss.item())
if (epoch+1) % 100 == 0:
    print(f"Epoch {epoch+1}/{n_epochs}, Train Loss: {loss.item():.4f}, Test Loss: {test_loss.item():.4f}")

# Plot the train and test losses
plt.plot(train_losses, label='Train Loss')
plt.plot(test_losses, label='Test Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Train and Test Losses')
plt.legend()
plt.show()
```

Epoch 100/1000, Train Loss: 10427.5957, Test Loss: 11847.3701  
Epoch 200/1000, Train Loss: 5962.8159, Test Loss: 6308.5332  
Epoch 300/1000, Train Loss: 4116.4946, Test Loss: 3035.7437  
Epoch 400/1000, Train Loss: 2273.3057, Test Loss: 2284.0188  
Epoch 500/1000, Train Loss: 2239.5327, Test Loss: 2025.8719  
Epoch 600/1000, Train Loss: 1257.2456, Test Loss: 1789.8909  
Epoch 700/1000, Train Loss: 1347.9012, Test Loss: 1598.5265  
Epoch 800/1000, Train Loss: 1585.1932, Test Loss: 1421.0337  
Epoch 900/1000, Train Loss: 1214.2162, Test Loss: 1277.1624  
Epoch 1000/1000, Train Loss: 1133.4139, Test Loss: 1143.3744

Train and Test Losses



Revision #8

Created 25 April 2023 02:44:34 by Loki Ng

Updated 5 May 2023 15:21:58 by Loki Ng